CASIO

再生紙を使用しています。

スーパーカレッジ VX-4 言語解説書

CASIO

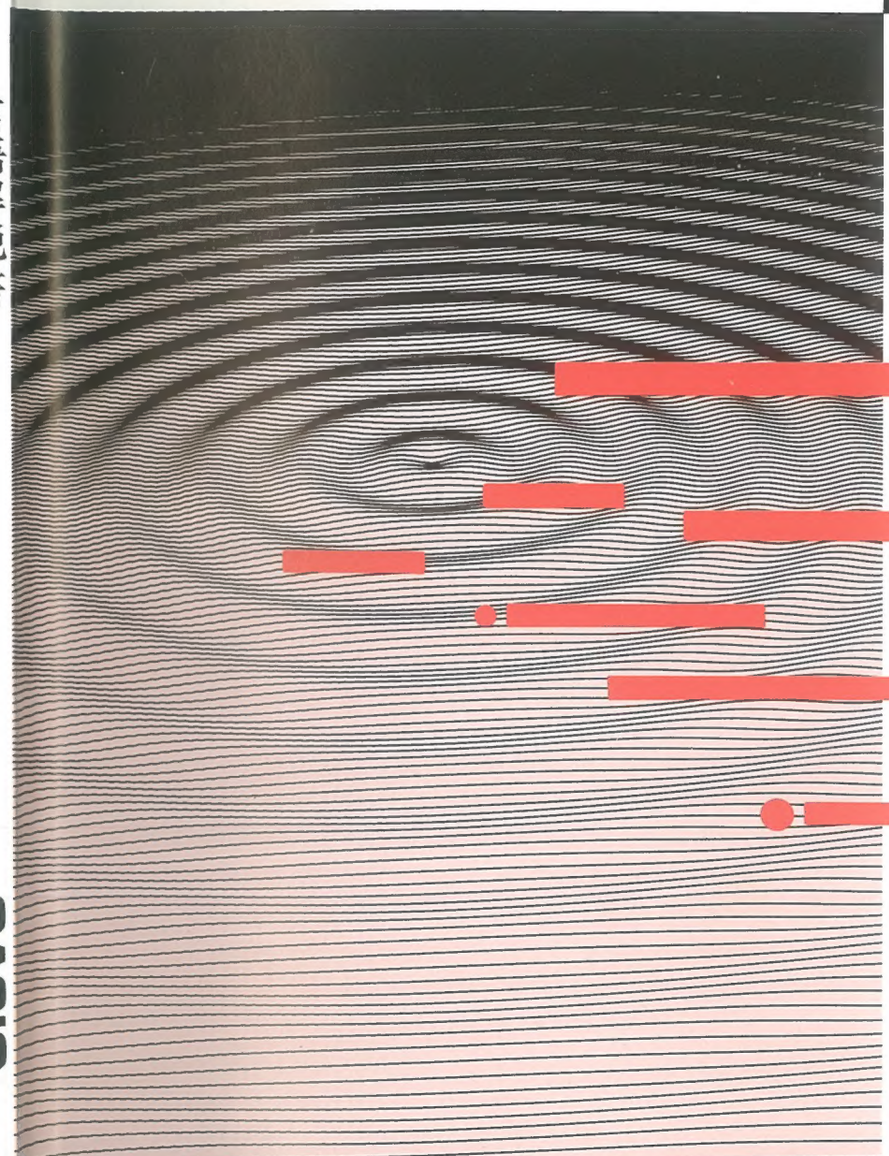
2004年12月現在　MO0412-006031P

スーパーカレッジ

VX-4

言語解説書

C・BASIC・CASL



CASIO

- 本書の内容に関しては、将来予告なしに変更することがあります。
- 本書の内容については万全を期して作成いたしましたが、万一不審な点や誤りなど、お気付きのことがありましたらご連絡ください。
- 本書使用による損害および逸失利益等につきましては、当社では一切その責任を負えませんので、あらかじめご了承ください。
- 本書の一部または全部を無断で複写することは禁止されています。また、個人としてご利用になるほかは、著作権法上、当社に無断では使用できませんのでご注意ください。
- 万一、本機使用により生じた損害、逸失利益または第三者からのいかなる請求についても、当社では一切その責任を負えませんので、あらかじめご了承ください。
- 故障、修理、電池交換等に起因するデータ・プログラムの消失による、損害および逸失利益等につきましては、当社では一切その責任を負えませんので、あらかじめご了承ください。
  なお、大切なデータおよびプログラムはフロッピーディスクやカセットテープに記録しておいたり、また控えのフロッピーディスクやカセットテープを作っておくようにしてください。

はじめに

工学系を目指す人の基礎力として、自分でコンピュータプログラムが作れることは大切なことです。

コンピュータ言語を学ぶには、文法解説書や専門書を読んで学ぶ方法があります。しかし、この方法では実際にコンピュータが動いているという実感をつかめず、多くの人がプログラムを作れるようになる前に興味を失ってしまいます。

プログラムの学習には近道はありません。プログラムの学習は「習うより慣れろ」という言葉がぴったりと当てはまります。コンピュータにプログラムを入力し、実行して結果を見ると、初心者にもコンピュータはこういうふうに動くということを実感できます。また、プログラムを改造して結果を得たり、自分で作ったプログラムを実行してデバッグしたりしながら学ぶという方法が、いちばん身につく学習方法なのです。

本機にはC言語、BASIC、CASLの３つのプログラム言語が搭載されています。

Ｃ言語は、いま最も注目されているプログラム言語です。プログラムを書くときの自由度が高い高級言語です。また、アセンブラのように直接メモリーを扱うことができて細かい処理も記述できます。科学技術計算だけでなくコンピュータによる制御のプログラムを書くのにも適しています。これは、Ｃ言語の特長のほんの一つの例です。将来、コンピュータを扱うさまざまな分野でＣ言語プログラマーが活躍すると予想されています。

BASICは、学習用のプログラム言語として、あるいは実験や研究の道具として広く使われています。本機のBASICは、特に数値計算の分野では高速で精度の高い計算ができます。学習用としてだけでなく、将来実務についたときも仕事の中で必ず役立てることができます。

CASLは、通産省の実施する「情報処理技術者試験」のためのアセンブリ言語です。言語仕様が限定されているので、初心者に理解しやすくコンピュータの動きを学ぶのには、最適なアセンブリ言語です。本機では、仮想計算機COMETとアセンブリ言語をシミュレートします。

本機本体だけで、簡単なキー操作を行なうことにより、プログラムを作成して実行させることができます。教室でも、自分の机の上でも、公園のベンチでも、電車の中でもどこでも気軽に言語体験ができます。どんな言語解説書を読むよりも、プログラムを作れるようになる近道となります。いつも本機を身近に置いてご活用いただきたいと思います。

目次

はじめに ——— III
目次 ——— IV
C言語・CASLのエディタ ——— VI

第1章 C言語編

簡単なプログラムとその実行 ——— 2
文字と数値を出力するプログラム ——— 5
文字と数値を入力するプログラム ——— 9
簡単な計算をするプログラム ——— 14
条件分岐を行なうプログラム ——— 18
繰り返し処理のプログラム ——— 22
配列とポインタ ——— 26
関数定義 ——— 30
記憶クラス ——— 34
文法のまとめ ——— 36
マニュアルコマンドリファレンス ——— 54
ライブラリー関数リファレンス ——— 55

第2章 BASIC編

プログラムの構成 ——— 76
インタープリターの操作 ——— 78
プログラムの入力と実行 ——— 80
エディタの操作 ——— 82
条件分岐するプログラム ——— 86
繰り返しのプログラム ——— 89
無条件ジャンプのプログラム ——— 91
サブルーチンの使い方 ——— 93
READ文とDATA文 ——— 95
配列変数の宣言 ——— 97
周辺機器との入出力 ——— 100
文法のまとめ ——— 104
マニュアルコマンド一覧表 ——— 106
プログラムコマンド一覧表 ——— 109

第3章 CASL編

簡単なプログラム ——— 116
ソースプログラムの編集・訂正・アセンブル ——— 118
オブジェクトプログラム ——— 119
エラーメッセージ ——— 121
割り算のプログラム ——— 122
プログラムのダンプ ——— 123
プログラムのトレース ——— 125
操作のまとめ ——— 127
CASLの紹介 ——— 128
COMETの構成 ——— 129
データの構成とレジスタ ——— 131
命令語の構成 ——— 133
アドレス修飾と実効アドレス ——— 134
スタック ——— 135
キャラクターセット ——— 137
オペレーティングシステム ——— 139
命令の形式 ——— 140
命令の種類と表記法 ——— 142
メモリーマップ ——— 156
C言語エラーメッセージ一覧表 ——— 157
索引 ——— 161

C言語・CASLのエディタ

ここで学ぶこと
☆ファイルエディタの使い方

まとめ
◎C言語とCASLのプログラム作成・修正は、同じ手順で行ないます。
◎新しいプログラムは、1行ごとにEXEキーを押します。
◎SEARCH、SHIFT＋LINEキーで、行、文字(列)の検索ができます。

ファイルエディタ

C言語とCASLのプログラムを作成したり修正したりする作業を「プログラムの編集」といい、編集を行なう機能をエディタといいます。また、作成したプログラム(ソースプログラム)は、本機のF0～F9のファイルエリアに記憶されます。

そこで、C言語とCASLのプログラムを編集するための機能、すなわちC言語のエディタと、CASLのエディタを総称してファイルエディタといいます。

本機を使ってC言語やCASLを学ぶには、ファイルエディタの操作に習熟することが大切です。

ファイルエディタの使い方

ファイルエディタの起動

プログラムを作成したり修正したりする場合には、まずファイルエディタを起動します。エディタ起動方法には、次の3通りがあります。

①C言語またはCASLのメニュー画面で、Sキーを押してプログラム入力状態にします。
②C言語のプログラム実行画面で、マニュアルコマンドEDITを入力します。
③F.COMメニューから編集するファイルエリアをカーソルキー(↑↓←→)で指定した後、Eキーを押します。

挿入モードと上書きモード

ファイルエディタでは、CALモードおよびBASICモードとは異なり、文字を入力するとカーソルの位置に文字が挿入され、それ以降の文字は後に送られます。この状態を「挿入モード(インサートモード)」といいます。

挿入モードでINSキーを押すと、「上書きモード(オーバーライトモード)」に変わります。この上書きモードでは、文字を入力するとカーソルの位置に文字が入力され、それまでカーソル位置にあった文字が消えます。

「上書きモード」でINSキーを押すと、挿入モードに戻ります。このように、INSキーを押すと、

挿入モード→上書きモード→挿入モード→上書きモード→……

のように順次切り替わります。

カーソルの形

ファイルエディタでは、カーソルは次のような形になります。

挿入モード				上書きモード			
英字	英字シフト	カナ	カナシフト	英字	英字シフト	カナ	カナシフト
■	▮	▬	▪	＿	＿	▬	▪

ファイルエディタの操作

エディタの起動とソースプログラムの入力

C言語	CASL
操作　ON SHIFT ©	操作　ON CASL
(C) F 0 1 2 3 4 5 6 7 8 9　3355B F2>Run/Load/Source/Cal	(CASL) F 0 1 2 3 4 5 6 7 8 9　3355B F1>Assemble/Source/Cal
メニュー画面になります。 操作　S	メニュー画面になります。 操作　S
■ (1)	■ (1)
ファイルエディタに入ります。 英小文字モードになっています。 操作　M A I N ( )	ファイルエディタに入ります。 英大文字モードになっています。 操作　B G N TAB S T A R T
main()■ (1)	BGN　START■ (1)
1行目のプログラムが入力できました。 操作　EXE	1行目のプログラムが入力できました。 操作　EXE
main() ■ (2)	BGN　START ■ (2)
カーソルが2行目に移動します。	カーソルが2行目に移動します。

プログラムの入力

プログラムは、入力した文字がそのままファイルエリアに記憶されます。改行する場合は、EXEキーを押します。

すべての入力が終わったら、BRKキーを押してCALモードに戻るか、SHIFT © あるいはCASLを押してファイルエディタ以外のモードに脱出してください。

プログラムの修正

まず、修正したい箇所に⇧⇩⇦⇨キーでカーソルを移動します。修正の操作方法には、次の4通りがあります。

操作

①挿入する文字を入力……文字を挿入します。改行したいときはEXEキーを押します。
②SHIFT＋DEL……文字を削除します。
③INS……キーを押して上書きモードにします。正しい文字を入力します。
④SHIFT＋LCAN……1行全部を削除します。

修正が終わったら、BRKキーを押します。

カーソルの移動

ファイルエディタでは、作成したプログラムの先頭から末尾までのすべてをスクロールさせて見ることができます。
カーソルの移動には、⇧⇩⇦⇨キーを使う以外に次の操作ができます。

操作

①SHIFT＋LTOP……カーソルのある行の行頭に移動します。
②SHIFT＋LEND……カーソルのある行の行末に移動します。
③SHIFT＋F.TOP……ファイルの先頭に移動します。
④SHIFT＋F.END……ファイルの末尾に移動します。

プログラムの中の文字(列)、行番号の検索

プログラムの中の文字(列)や行番号による検索ができます。プログラムの修正に便利な機能です。ここでは、C言語のプログラムを例にして説明します。

文字列「in」を検索する場合

操作 SEARCH

```
main()
{
      printf("コンニチハ!");
search?_                          ( 1)
```

操作 I N EXE
(検索したい文字列を入力)

```
main()
{
      printf("コンニチハ!");
                                  ( 1)
```

操作 NEXT(次の「in」にカーソルが移動します)

```
      printf("コンニチハ!");
}
                                  ( 3)
```


行番号3を検索する場合

操作 SHIFT LINE

```
main()
{
      printf("コンニチハ!");
line?_                            ( 1)
```

操作 3 EXE
(検索したい行番号を入力)

```
■     printf("コンニチハ!");
}
                                  ( 3)
```

3行目が表示されます。

ファイルエディタを使うときの注意

ファイルエディタによりF0～F9のそれぞれのファイルを編集できます。その際、1行で256文字以上のデータを扱えます。しかしBASICやデータバンクでは1行で扱える文字数が255文字までなので、1行に256文字以上の文字データを扱う場合は、次のような注意が必要です。

またファイルエディタでは、EXEキーを押さなくてもデータが入力できますが、このようなデータが最後にあるようなファイルをBASICやデータバンクで扱うときも同様の注意が必要です。

①F0エリアにこのようなデータがある場合
MEMおよびMODE 9と操作しないでください。誤ってMODE 9(メモインモード)と操作すると、BV errorが発生します。こうなったときは、SHIFT F.COMと操作してください。
②F0～F9のエリアにこのようなデータがある場合
READ#、WRITE#などのデータバンク用命令を実行しないでください。
③I/Oにこのようなデータがある場合
BASICエリアにロードできません。
④このようなデータはBASICエリアにファイルコピーをしないでください。
注) I/Oとは、入出力機器を意味します。

第1章
C言語編

簡単なプログラムとその実行

ここで学ぶこと
☆C言語モード
☆C言語モードのメニュー

まとめ
◎メニュー画面から、次のキーを押します。
[S]キー:プログラムの入力・編集
[L]キー:プログラムのロード
[R]キー:プログラムの実行

〔プログラム例〕 あいさつのプログラム

```
①  main()
②  {
③      printf("コンニチハ");
④  }
```

C言語のプログラムは、特別な場合を除いて英小文字で記述します。また、カッコを多用するのも特徴です。

C言語モード

まず、電源をONにして、C言語モードに入ります。

操作		表示	
①	ON SHIFT C	①	＜ C ＞ F 0 1 2 3 4 5 6 7 8 9 33558 F2>Run/Load/Source/Cal

画面の説明

この画面が、C言語モードのメニュー画面です。

3行目の数字は、ファイルエリアの番号を示しています。ファイルエリアにプログラムなどが書き込まれていると、そのファイルエリアの数字の代わりに「＊」(アスタリスク)が表示されます。何も書き込まれていないファイルエリアの番号に、[←][→]でカーソルを移動します。

4行目には、これから使用するファイルエリアの番号と操作のメニューが表示されています。メニューの頭文字を入力して機能を選択します。

メニュー	キー	機　　能
Source	[S]キー	プログラムの入力と修正が行なえます。
Load	[L]キー	プログラムがロードされます。
Run	[R]キー	プログラムが実行されます。
Cal	[C]キー	マニュアル計算モードに戻ります。

プログラムの入力

まず、ファイルエリアの番号を選びます(この操作例では、「2」を選びます)。次に、プログラムを入力するために[S]キーを押します。プログラムは次のように入力します。

操作		表示	
①	M A I N ( ) EXE	①	main() {
②	SHIFT (EXE	②	
③	TAB P R I N T F (SHIFT " SHIFT カナ K O N N I T I H A SHIFT カナ SHIFT ") SHIFT ; EXE	③	printf("コンニチハ"); }
④	SHIFT) EXE	④	_

プログラムを入力し終ったら、SHIFT C とキーを押して、もう一度先ほどのC言語のメニュー画面に戻ってください。

プログラムのロード

操作		表示	
①	[L]	①	Load F2 >_

「F2」エリアのプログラムがロードされて、入力待ちの状態になりました。

プログラムの実行

操作		表示	
①	[R][U][N] EXE	①	>run コンニチハ >_

「run」あるいは「RUN」と入力すると、結果が表示されます。また、SHIFT RUN EXE という操作でも同様です。

メニューからの実行

もう一度先ほどのメニュー画面に戻ってください。

操作		表示	
①	R	①	Load F2 コンニチハ >ー

メニューからRキーを押すと、プログラムがロードされ、続けてプログラムが実行されます。

プログラムの編集

メニュー画面からすでにプログラムが書き込まれているエリアを指定してSキーを押すと、プログラムの編集画面になります。プログラムを書き直すときには、この操作を行なってください。

エラーとプログラムの修正

エラーが発生すると、エラーの発生した行番号とエラーメッセージが表示されます。

そこでEDITコマンドを入力すると、エラーの発生した行が表示されるので、プログラムを確認して修正してください。

文字と数値を出力するプログラム

ここで学ぶこと
- ☆コメントの書き方
- ☆main( )関数
- ☆；と｛｝
- ☆変数の型宣言
- ☆変数への代入
- ☆printf関数
- ☆￥n（改行）
- ☆%文字、エスケープシーケンス
- ☆データの編集形式と種類
- ☆代表的なエスケープシーケンス
- ☆データの表示

文法のまとめ
- ◎コメントは、プログラム中のどこにでも書けます。
- ◎プログラムの実行は、必ずmain( )からはじまります。
- ◎文の終わりには；をつけ、まとまった文は｛と｝でかこみます。
- ◎使用する変数は、型宣言します。
- ◎変数にはプログラムの中で数値などを代入できます。
- ◎文字や数字を表示するには、printf関数を使います。
- ◎%文字はprint関数の書式で、データの表示位置と編集形式を指定します。
- ◎エスケープシーケンスは、特別な意味を持つ文字定数です。printf関数の書式で使用します。
- ◎改行には、￥nを使います。

〔プログラム例〕 文字や数値を表示するプログラム

```
①  /* モジ゛ ヤ スウジ゛ ヲ ヒョウジ゛ スル フ゜ロク゛ラム */
②  main()
③  {
④       char   moji;
④       int    suuji;
⑤       moji='A';
⑤       suuji=123;
⑥       printf("モ シ゛=%c¥n", moji);
⑥       printf("スウジ゛=%d¥n", suuji);
③  }
```

C言語のプログラムの特長

①コメント（注釈）

プログラムのタイトルなどをプログラム中にコメントとして書き込むことができます。コメントは、／＊と＊／で囲みます。

行の先頭に／＊、間に文字や記号、最後に＊／を書いて終了するとその全体がコメント行になります。

コメントはプログラムの実行に影響しません。コメント中には、なにを書いてもエラーになりません。

②main関数

C言語のプログラムの冒頭には、必ずmain( )を書きます。main( )は主プログラムであることを示す特別な関数です。C言語のプログラムはそれ自体が関数の集まりであり、関数を定義したり、関数を組み合わせて構成します。

③プログラムの最小単位「文」

プログラムの最小単位が文です。1つの文(単文)の終わりには必ず「;」(セミコロン)を付けます。単文がいくつか集まったものを複文(またはブロック)といい、複文は { と } で囲みます。

④変数の型宣言

プログラムで使用するすべての変数は、プログラムのはじめに宣言します。宣言する内容は、変数の名前とその変数の型です。

ここでは、mojiという名前の変数と、suujiという名前の変数を宣言しています。

```
char moji;  char型(文字型)の変数宣言
int  suuji; int型(整数型)の変数宣言
```

変数の型については、後の「簡単な計算をするプログラム」で学びます。

⑤データの代入

変数間のデータの代入には式を使います。式の右辺の内容が、式の左辺の変数に格納されます。

ここでは、変数mojiにAが、変数suujiに123が代入されています。なお、「A」の前後につけられた「'」と「'」(シングルクォーテーション)は「A」という文字であること（文字定数であること）を示しています。

コラム●コメントは自由に書ける●

次の例のように、コメントは関数を書いた行にも書くことができます。

```
printf("モ ジ=%c¥n",moji); /* moji ヲ ヒョウジ スル */
```

また、複数の行にまたがるコメントを書くこともできます。

```
/*********************************/
/*        ゲーム ノ プログラム        */
/*    (C)KOTAROU, February 1990    */
/*********************************/
```

ただし、/*/* コメント */*/のようにコメントの中にコメントは書けません。

文字や数値を出力する関数

⑥printf 関数

プログラムの⑥をつけた2行は、両方ともprintf関数による文です。printf関数はデータを表示させる基本的な関数で、C言語ではもっともよく使われる関数の一つです。

printfに続く( )の中身を引数といいます。引数が複数の場合は「,」(カンマ)で区切ります。この場合は引数は2つあります。2つの引数の意味は次のようになっています。

```
printf("モ ジ=%c¥n", moji);
       書式          変数
printf("スウジ=%d¥n", suuji);
       書式           変数
```

printf関数は、上の書式で指定した形で変数の部分のデータを変換して出力するようになっています。

書式の部分に文字列を書くと、そのとおりに表示されます。もちろん文字列を書かなければ表示されません。ここでは「モジ=」と「スウジ=」の部分が表示されます。

変数の内容を表示したい場所に「%文字」を書きます。%dは整数(10進数)を表示するとき、%cは1文字を表示するときに書きます。主な%文字を次に示します。

%文字	変換結果	プログラム例⑥で、moji='A'を変換して出力すると
%d 注1)	整数(10進数)	65
%f	実数(10進数)	注2)
%c	1文字	A
%s	文字列	注3)
%x	整数(16進数)	41

注1) 桁の大きい整数(long型)を扱うときは、%ldを使います。
注2) %fでは文字の変換はできません。
注3) この例では変数mojiは1文字として定義されていますので、%sの指定はできません。

このように、%d、%xを指定すると、「'A'」がASCIIコードに変換されます。

エスケープシーケンス

エスケープシーケンスとは特別な文字定数で、printf関数の書式で使用します。主に改行やブザーを鳴らしたい場合に使用します。よく使うエスケープシーケンスを次に示します。(その他のエスケープシーケンスについては、「文法のまとめ」を参照してください。)

表記	呼び方(略名)	意　味
¥n	New Line (NL)	改行(+復帰)
¥f	Form Feed	改ページ
¥a	Bell (BEL)	ブザーを鳴らす

コラム●printf関数で複数の表示を行なう●

2つ以上の変数の内容を表示するときには次のようにします。

```
printf("モ　ジ=%c スウジ=%d ¥n", moji, suuji);
```

①複数の変数の内容を表示する位置を指定します。
②表示位置に対応する変数名を指定します。
③最初の%cの位置にはmoji、次の%cの位置にはsuujiの内容を表示します。

文字と数値を入力するプログラム

ここで学ぶこと
☆scanf関数
☆整数定数
☆浮動小数点定数
☆文字列定数
☆文字定数

文法のまとめ
◎scanf関数は文字や数値データを入力します。
◎数値定数は整数定数と浮動小数点定数があります。
◎整数定数は、10進定数、8進定数、16進定数があります。
◎文字定数には、文字定数、文字列定数があります。

〔プログラム例〕 文字や数値を入力するプログラム

```
main()
{
        char    moji;
        int     suuji;
        printf("1モジ ヲ ニュウリョク シテ クダサイ=");
①       scanf("%c", &moji);
        printf("スウジ ヲ ニュウリョク シテ クダサイ=");
②       scanf("%d", &suuji);
        printf("1モジ=%c¥n", moji);
        printf("スウジ=%d¥n", suuji);
}
```

文字や数値を入力する関数

①scanf関数——文字の入力

scanf関数はデータを読み込む機能があります。次のような引数を記述します。

```
scanf("%c", &moji);
      (書式) (変数)
```

書式:%文字は入力するデータの形式を指定します。
%cは1文字を入力するときに書きます。
変数:読み込んだデータを代入する変数の名前を書きます。また、変数名の前には単純変数の場合、&を書きます。

プログラムの①の「&moji」のように、変数の前に&を付けます。&は番地(アドレス)を表わす記号で、&mojiは変数mojiの格納されるメモリーの番地(アドレス)を表わします。このように、scanf関数では、データを格納する変数の前には&を付けます。

書式の「%c」は1文字を入力することを示します。入力されたデータが「1文字としてのデータである」として扱います。すなわち、「1」と入力しても文字としての1が入力されたのであり、数値としての1ではないと判断されます。

②scanf関数——数値の入力

プログラムの②では数値を入力します。入力された数値データを格納する変数は、int型、long型、float型、double型のいずれかで宣言しておきます。ここではint型を宣言しています。

%dは、数値データを入力することを示す書式です。int型に対応しています。

変数suujiの前には、格納されるメモリーの番地を示す&をつけます。

ここで入力されるデータは、int型(整数型)としての数値となります。例えば、「1.23」と入力しても、整数部分の「1」だけが認識されて、小数部分は無視されます。

数値の種類

5ページのプログラム例の「suuji＝123;」の「123」のようなデータのことを定数といいます。定数にはいくつかの種類があり、「123」は整数定数の10進定数です。C言語では、このほか8進数(8進定数)や16進数(16進定数)を扱うことができます。
また、小数点数（浮動小数点定数）も扱えます。

```
        ：
int  suu1, suu2, suu3 ;
/*  セイスウ  ノ  ヘンスウ  ヲ  シテイ  スル  */
float  suuf ;
/*  シ゛ッスウ  ノ  ヘンスウ  ヲ  シテイ  スル  */
③ suu1＝12 ;
/*  セイスウ  12  ヲ  10シン  ケイシキ  テ゛  アタエル  */

④ suu2＝014 ;
/*  セイスウ  12  ヲ   8シン  ケイシキ  テ゛  アタエル  */

⑤ suu3＝0xc ;
/*  セイスウ  12  ヲ  16シン  ケイシキ  テ゛  アタエル  */

⑥ suuf＝3．1416
/*  シ゛ッスウ  3．1416  ヲ  10シン  ケイシキ  テ゛  アタエル  */
        ：
```

③10進定数

整数の数値を10進数で与えます。特に指定しないと10進数になります。

④8進定数

整数の数値を8進数で与えます。0(ゼロ)から始まる数値は、8進数になります。

⑤16進定数

整数の数値を16進数で与えます。0x(ゼロエックス)から始まる数値は、16進数になります。

⑥浮動小数点定数（実数）

実数の数値は、浮動小数点の10進数で与えます。

コラム●複数のデータの入力●

2つ以上の変数を入力するときには次のようにします。

```
scanf"("%c%d", &moji, &suuji);
```

複数のデータを入力する%文字を指定します。

%文字に対応する変数名を指定します。入力されたデータを%文字に応じて変換し、指定された変数へ代入します。%cが&mojiに、%dが&suujiに対応しています。

文字の種類

数値のほかに、文字も定数の一種として扱えます。文字を扱うには次のようにします。

```
                ⋮
char  ichimoji;
/*  1モジ゛ノ  ヘンスウ  ヲ  シテイ  スル  */
char  mojiretu[6];
/*  モジ゛レツ  ノ  ヘンスウ  ヲ  シテイ  スル  */
⑦  ichimoji='A';
/*  1モジ゛  A  ヲ  アタエル  */
⑧  strcopy(mojiretu,"ABCDE");
/*  モジ゛レツ  ABCDE  ヲ  アタエル  */
                ⋮
```

⑦文字定数

文字を与えます。データを「'」(シングルクォーテーション) で囲むと１文字形式の文字定数になります。

⑧文字列定数

文字列を与えます。データを「"」(ダブルクォーテーション)で囲むと文字列定数になります。

文字列の最後にはデータの終わりを表わす「¥0」(NULL)が自動的に付加されます。

定数の種類

種　類	形　式	例
数値タイプ		
整数定数	10進定数	int型　12，−12，32767 long型　123456789L
	８進定数	int型　014，0777 long型　012345L
	16進定数	int型　0xB，0xF4，0xFFFF long型　0x7FFFFFFFL
実数定数	浮動小数点定数	3.1416，−12.0，1.0e−5，5.555e10
文字タイプ		
１文字定数	一般の１文字定数	'A','a','1','0'
	エスケープシーケンス	'¥n'(改行)，'¥a'(ベル) '¥¥'(¥という文字) '¥''('という文字) '¥"'("という文字)
文字列定数		"コンニチハ","ABCDE","abcde","123456" "1モジ　ニュウリョク　シテ　シダサイ"

〔いろいろなタイプを扱ってみるプログラム〕

```
main()
{
        int  suui;
/*  セイスウ  ノ  ヘンスウ  ヲ  シテイ  スル  */
        float  suuf;
/*  ジ゛ッスウ  ノ  ヘンスウ  ヲ  シテイ  スル  */
        char  ichimoji;
/*  1モジ゛ノ  ヘンスウ  ヲ  シテイ  スル  */
        char  mojiretu[4];
/*  モジ゛レツ  ノ  ヘンスウ  ヲ  シテイ  スル  */
        printf("セイスウチ  ヲ  ニュウリョク  シテ  クダ゛サイ=");注1)
⑨      scanf("%d",&suui);
⑨      printf("スウチ=%d(10シン)  %o(8シン)
  %x(16シン)¥n",suui,suui,suui);注1)
        printf("スウチ  ヲ  ニュウリョク  シテ  クダ゛サイ=");注1)
        scanf("%f",&suuf);
        printf("スウチ=%f(フト゛ウショウスウテン)¥
n",suuf);注1)
        printf("1モジ゛  ヲ  ニュウリョク  シテ  クダ゛
サイ=");注1)
        scanf("%c",&ichimoji);
        scanf("%c",&ichimoji);
        printf("1モジ゛=%c¥n",ichimoji);
        printf("モジ゛レツ(3モジ゛)  ヲ  ニュウリョク
シテ  クダ゛サイ=");注1)
⑩      scanf("%s¥n",mojiretu);注3)
⑩      printf("モジ゛レツ=%s",mojiretu);注3)
}
```

注1) 下線部は、1行として入力してください。

注2) 配列はそれ自体がメモリーの番地(アドレス)を表わしますので、&は不要です。

注3) ¥n(改行)指定は、scanfでの入力の最後に押されるEXEキーがその働きをしますから不要です。
文字列の最後にはデータの終わりを表わす「¥0」(NULL)が自動的に付加されます。

⑨10進から８進、16進への変換

入力した数値を、%文字の指定するかたちに変換して出力します。ここでは、「suuji」という変数の内容を10進、８進、16進の３つの表現で出力します。

「%o」は８進、「%x」は16進になります。

⑩文字列の入出力

文字列は、配列という機能を使ってメモリーに記憶されます。配列については後で解説します。

簡単な計算をするプログラム

ここで学ぶこと
☆変数名の付け方
☆変数の型宣言
☆データ型の種類
☆算術演算子

文法のまとめ
◎int型は16ビットの整数。
◎代入演算子で変数に数値を代入します。
◎long型は32ビットの整数。
◎float型は32ビットの実数(浮動小数点数)。
◎double型は64ビットの実数（倍精度浮動小数点数)。

〔プログラム例〕 文字や数値を表示するプログラム

```
/*   ヘンスウ ノ カタセンケﾞン   */
main()
{
①       int         suu1, suu2, suu3;
/* ミジﾞカイ ケタ ノ セイスウ */

②       suu1=6;
②       suu2=4;

③       suu1=suu1+suu2;

③       suu3=suu1/2;

④       printf(" suu1+suu2=%d¥n", suu1);
④       printf(" suu1/2     =%d¥n", suu3);
}
```

変数名

C言語では、プログラムで使用するすべての変数を、プログラムの最初に宣言します。また、変数の型も同時に宣言します。

変数名は、次の規則の範囲で自由に付けられます。

変数名の規則

- 英字で始まる文字列と数字を組み合せてください。
- _(下線)は、英文として使用できます。
- 長さに制限はありませんが、先頭から8文字だけが有効です。
- キーワードだけの変数名は使えません。
- ライブラリー関数名とキーワードは使えません。

- キーワードは、「C言語のまとめ」の「キーワード」のページで説明します。
- 普通、変数名は小文字を使います。

変数の型宣言

変数の型には、次のような種類があります。必ずどれかを宣言しなければなりません。

データ型	扱うデータ	ビット数
char	1文字	8
int, long	整数	16, 32
float	浮動小数点数	32
double	倍精度浮動小数点数	64

①int型の宣言

int型は、整数を扱う変数の型です。int型変数の扱える数値は次の範囲です。

$-32768 \leqq n \leqq 32767$

2つ以上の変数名を続けて宣言することもできます。このときは、「,」(カンマ)で区切って書きます。

算術演算子と代入演算子

計算に使用する演算子を、算術演算子といいます。算術演算子には次の種類があります。

演算子	意　味	例[1]	演算内容
＋	加算	suu1＝suu1＋suu2; [2]	10＝6＋4
－	減算	suu1＝suu1－suu2;	2＝6－4
＊	乗算	suu3＝suu1＊suu2;	24＝6＊4
／	除算[3]	suu3＝suu1／suu2;	1＝6／4
％	剰余[4]	suu3＝suu1％suu2;	2＝6％4

注1) suu1、suu2はともにint型で、suu1＝6、suu2＝4とします。
注2) 例で使用している＝は、代入演算子の一種です。
注3) 変数が整数どうしの除算(割り算)では、結果の小数部分が切り捨てられます。
注4) 剰余(余り)を求めるときには、％を使います。「％文字」と同じ記号ですが意味はまったく違いますので気を付けてください。

②代入演算子

＝は、代入演算子です。右辺の結果を左辺に代入します。suu1には数値6が、suu2には数値4が代入されました。

③加算・乗算と代入

上の表の算術演算子を、プログラムの中に書いて計算させることができます。ここでは、加算と乗算の算術演算子を使いました。

算術演算の結果を、代入演算子＝を使って代入しています。このように右辺と左辺に同じ変数があっても構いません。

suu1＝6，suu2＝4 ですから計算結果は、次のようになります。

```
suu1＝suu1＋suu2    :   10＝6＋4
suu3＝suu1/2        :    5＝10/2
```

〔プログラム例〕 大きな数、小さな数

```
main()
{
⑤       long        suul, suum, suun;
/*  ナカ゛イ   ケタ  ノ  セイスウ  */
⑥       float       suuf, suug, suuh;
/*  シ゛ッスウ(フト゛ウ  ショウスウテンスウ)  */
⑦       double      suud, suuc, suub;
/*  スコ゛ク  ナカ゛イ  ケタ  ノ  シ゛ッスウ          */
/*  (ハ゛イセイト゛  フト゛ウ  ショウスウテンスウ)  */

⑧       suul=32768; suum=100000;
⑧       suuf=0.000001; suug=0.00001;
⑧       suud=1.23e+70; suuc=1.23;


⑨       suun=suul+suum;
⑨       suuh=suuf-suug;
⑨       suub=suud/suuc;

⑩       printf("(long)     %ld¥n", suun);
⑩       printf("(float)    %f¥n", suuh);
⑩       printf("(double)   %lf¥n", suub);
}
```

大きな数、小さな数

⑤long型の宣言

long型は、int型と同じように整数を扱う変数の型です。ただしlong型は、int型より大きい数値を扱えます。範囲は次のようになります。

$-2147483648 \leqq n \leqq 2147483647$

⑥float型の宣言

float型は、実数(浮動小数点数)を扱う変数の型です。数値に小数桁が必要なときに使います。

float型変数の扱える数値は次の範囲です。

0 および ±1e−63 ～ ±9.99999e+63

⑦double型の宣言

double型は、float型と同じように実数を扱う変数の型です。ただしdouble型は、float型より大きい数値を扱えます。範囲は次の通りです。

0 および ±1e−99 ～ ±9.9999999999e+99

⑩結果出力

printf関数を使って、suul、suuh、suubに代入された数値を出力しています。それぞれの数値の型がわかるように(long)、(float)、(double)の文字列もいっしょに出力するようにしてあります。

コラム●桁数の整数や実数を扱うときのプログラム●

```
main()
{
      int sua, sub;
      long sul;
      float suf;

      sua=700;
      sub=250;

      sul=sua+sub;
      suf=sul/2.0;  /*  チュウ1)  */

      printf("sua=%5d, sub=%5d¥n", sua, sub);
      /*  チュウ2)  */
      printf("sul=%8ld, suf=%8.2f¥n", sul, suf);
      /*  チュウ3)  */
}
```

①「suf＝sul/2.0;」のように演算式に実数を書くと、実数の演算結果が得られます。

②「%5d」のように%文字を書くと、数値が編集されて出力されます。(この例では5桁です)

表示　　sua＝␣␣700,sub＝␣␣250　　(␣は空白の意味です)

③「%8.2f」のように%文字を書くと、数値が編集されて出力されます。小数点には「.」(ピリオド) を使い、小数点も1桁と数えます。

表示　　suf＝␣␣␣␣␣950,suf＝␣␣␣475.00　　(␣は空白の意味です)

コラム●char型の宣言●

このページで説明した数値データの型以外に、1文字を扱う変数の型として「char型」があります。これもよく使うものです。「文字と数値を出力するプログラム」で示したプログラムに使ってありますから参照してください。

条件分岐を行なうプログラム

ここで学ぶこと
☆ifによる分岐
☆比較演算子
☆論理演算子

文法のまとめ
◎if文が成立すると次に書かれた文が、成立しないとelseの次に書かれた文が実行されます。
◎比較演算子は、式の真偽によって0または1を返します。
◎論理演算子は、AND、OR、NOTの論理演算を行ないます。

〔プログラム例〕 偶数、奇数を判定するプログラム

```
/*  ｸﾞｳｽｳ, ｷｽｳ  ヲ  ﾊﾝﾃｲ  ｽﾙ  ﾌﾟﾛｸﾞﾗﾑ  */
main()
{
        int     suuji, hantei;
        printf("ｽｳｼﾞ  ヲ  ﾆｭｳﾘｮｸ  ｼﾃ  ｸﾀﾞｻｲ=");
        scanf("%d", &suuji);
        hantei=suuji%2;
/*  ﾆｭｳﾘｮｸ  ｼﾀ  ｽｳｼﾞヲ
        2ﾃﾞ  ﾜｯﾃ  ｱﾏﾘヲﾓﾄﾒﾙ  */
        if(hantei==0){
①               printf("%d  ﾊ  ｸﾞｳｽｳ  ﾃﾞｽ¥n",
②   suuji);注)
        }
        else{
③               printf("%d  ﾊ  ｷｽｳ  ﾃﾞｽ¥n",
    suuji);注)
        }
}
```

注) 下線部は1行として入力してください。

if文による分岐

①if文による条件の判定
変数hanteiの内容が0(ゼロ)のときは式が成立し、そうでないときは式が成立しません。

②式が成立するとき
式が成立するときには、このprintf関数だけが実行されます。

③式が成立しないとき
式が成立しないときには、このprintf関数だけが実行されます。

if文のまとめ

```
if(式){
      文1;
}
else{
      文2;
}
```

- 式の値が真（成立する）なら、文1を実行します。
- 式の値が偽（成立しない）なら、文2を実行します。
- もし不要な場合にはelse以下を全部省略します。
- 文1、文2が単文の場合には、{ }は不要です。

if文で指定できる条件には、次のものがあります。
<,>,<=,>=,==,!=

コラム●if文のもうひとつの書き方●

上記のif文を、次のように書くこともできます。

```
if(hantei==0){
      printf("%d  ﾊ  ｸﾞｳｽｳ  ﾃﾞｽ¥n", suuji);
}
else if(hantei!=0){
      printf("%d  ﾊ  ｷｽｳ  ﾃﾞｽ¥n", suuji);
}
```

比較演算子

このプログラムでは、「if (hantei＝＝0)」のように、等価比較を行なっています。この他、次のような演算子がif文には使えます。

比較演算子	意　味
i>j	iがjより大なら真、そうでないなら偽を与える
i<j	iがjより小なら真、そうでないなら偽を与える
i>=j	iがjより大または等しいなら真、そうでないなら偽を与える
i<=j	iがjより小または等しいなら真、そうでないなら偽を与える
i==j	iとjが等しいなら真、そうでないなら偽を与える
i!=j	iとjが等しくないなら真、そうでないなら偽を与える

コラム●if文の有効範囲●

｛｝の中のそれぞれ式が真のとき、偽のときの有効範囲です。if文のおよぶ範囲が一目でわかるように、｝はifのiと同じ桁に書き、文1、文2などはifよりも少し下げて書きます。これを字下げといいます。

論理演算子

先のプログラムの①を①'のように変えると、「suujiが100以下の場合で」という条件が付加されます。

〔プログラム例〕 偶数、奇数を判定するプログラム(suuji<＝100の条件付き)

```
①'  if(hantei==0 && suuji<=100){
②'      printf("%d ハ ク゛ウスウ テ゛ 100 イカ
    テ゛ス¥n", suuji); 注)
    }
    else{
③'      printf("%d ハ キスウ カ 101 イシ゛ョウ
    テ゛ス¥n", suuji); 注)
    }
```

注）下線部は1行として入力してください。

ここで使った&&は論理演算子のひとつで、ANDの演算を行ないます。論理演算子には、次のようなものがあります。

論理演算子	意　味	例
&&(AND)	2つの条件をANDで結びつける	if(hantei＝＝0 && suuji<＝100)
||(OR)	2つの条件をORで結びつける	if(hantei＝＝0 || suuji<＝100)
！(NOT)	2つの条件をNOTで結びつける	if(hantei！＝0)

&&を、||や！に変えると次のようになります。

&&(AND) ……AND条件

これでは、hanteiが0(ゼロ)で、suujiが100以下という条件になります。

||(OR) ……OR条件

これでは、hanteiが0または、suujiが100以下という条件になります。

！(NOT) ……NOT(否定)条件

この例では、hanteiが0以外という条件になります。

コラム●演算子の優先順位●

よく使う演算子を次に示します。演算子には、実行するときの優先順位があります。その他の演算子については「文法のまとめ」を参照してください。

		優先順位
算術演算子	* / + -	1
関係演算子	< <= > >=	2
比較演算子	== !=	3
論理演算子	&& ||	4
代入演算子	+= = etc	5

繰り返し処理のプログラム

ここで学ぶこと
☆while文による繰り返し
☆for文による繰り返し
☆インクリメント演算子、デクリメント演算子

文法のまとめ
◎while文は、条件式が真のあいだ、文を実行します。
◎for文は、条件式が真のあいだ、文を実行します。条件は初期値で与え、増分値で更新します。
◎インクリメント演算子、デクリメント演算子は、変数を＋１または－１します。

いろいろな繰り返し

```
/*  クリカエシ  ショリ  ノ  フ゜ロク゛ラム  */
main()
{
        int     suu1, suu2, i;
        suu1=-1;
①       while(suu1!=0){
/*  0  ヲ  ニュウリョク  スルト  シュウリョウ  シマス  */
        printf("スウシ゛  ヲ  ニュウリョク  シテクダ゛サイ=");注)
        scanf("%d", &suu1);
        suu2=suu1;
②③      for(i=1;i<=3;++i){
                suu2=suu2*suu2;
/*  ニョクリョク  サレタ  スウシ゛  ノ
    8  シ゛ョウ  ヲ  モトメマス  */
        }
        printf("%d  ノ  8  シ゛ョウ  ハ  %d
テ゛ス¥n", suu1, suu2);注)
    }
}
```

注）下線部は１行として入力してください。

①while文

繰り返し処理を指定するには、while文を使います。式で与える条件が成立しているあいだ、｛ ｝内の処理を繰り返します。while文の範囲は、｛ ｝で指定します。

②for文

繰り返し処理を指定するもうひとつの方法は、for文を使います。式で与える条件が成立しているあいだ、｛ ｝内の処理を繰り返します。for文の範囲は、｛ ｝で指定します。

③インクリメント演算子

インクリント演算子は、i=i＋１の演算をします。

while文のまとめ

while文は、指定した条件が成立するあいだ、処理を繰り返し実行します。

```
while(条件式) {
    文;
    :
}
```

繰り返し処理を行なう条件を式で指定します。条件が真（成立する）なら｛ ｝の中の処理を繰り返します。

while文の範囲は、｛ ｝で指定します。文が単文のときは、｛ ｝を省略できます。

プログラム例のfor文を、while文に書き換えると次のようになります。

```
i=1; /*ショキチ   */
while(i<=3){          /* ニョクリョク サレタ スウシ゛ */
  sun2=sun2*sun2;     /* 8 シ゛ョウ ヲ モトメマス */
  i=i+1;    /*ソ゛ウフ゛ンチ */
}
```

コラム●do～while文●

繰り返し処理は、このほかに、次のような種類があります。

```
do{
    文;
    :
} while(式);
```

コラム●無限ループ●

while文を無条件に繰り返す方法。（無限ループ）

```
while(1){
    :
}
```

無限ループから脱出する方法は、break文、return文、exit( )です。

forによる繰り返し

for文は、指定した条件が成立するあいだ、処理を繰り返し実行します。

```
for（初期値; 条件式; 増分式）｛
        ⋮
｝
```

繰り返し処理を行なう初期値、条件式、増分式を指定します。条件が真（成立する）なら｛｝の中の処理を繰り返します。

for文の範囲は、｛｝で指定します。文が単文のときは、｛｝を省略できます。

コラム●for文の変形●

for文は、２重３重のネスト（入れ子）にできます。

```
for(i=1;i<=3;++i){
    for(j=1+i;j<=3;++j){
        suu2=suu2*suu2;
    }
}
```

カンマ「,」で区切ることで、初期値や増分式を複数個指定できます。

```
for(i=1, j=10;i<j;++i, j--) {
        ⋮
}
```

for文を無条件に繰り返すには（無限ループ）は、初期値、条件式、増分式になにも書きません(空文とします)。

```
for(;;) {
        ⋮
}
```

無限ループから脱出する方法は、break文、return文、exit( )です。

インクリメント演算子とデクリメント演算子

＋＋をインクリメント演算子といいます。インクリメントは増分という意味です。

```
i=i+1;
```

と同じ動作をします。

－－をデクリメント演算子といいます。デクリメントは減分という意味です。

```
i=i-1;
```

と同じ動作をします。

プログラムの③の＋＋iはi=i+1と同じ意味になります。

コラム●j=++iとj=i++●

j=＋＋iとj=i＋＋では、実行結果が異なります。i= 3 の場合で考えてみましょう。

j=＋＋iという式では、iに1を加えてi=4としてから、jにiを代入します。したがって、j=4となります。

j=i＋＋という式では、iに1を加える前に、jにiを代入します。したがって、j=3となります。

また、if文などでは次のようになります。

```
if(＋＋i<3) ｛
        ⋮
｝
```

i=i+1の後、i<3の判定を行なう

インクリメント演算子、デクリメント演算子を、他の演算子と一緒に書くときには、気を付けなければなりません。

配列とポインタ

ここで学ぶこと
☆配列の宣言
☆配列へのデータの代入
☆ポインタ変数の宣言
☆＊(アスタリスク)演算子の使い方

文法のまとめ
◎配列を宣言するとメモリーに配列の領域が確保されます。
◎配列にはデータが代入できます。
◎「型名　＊変数名」でポインタ変数を宣言します。
◎ポインタ変数にはアドレスが格納されます。
◎＊演算子でアドレスの内容を参照します。

〔プログラム例〕　2つの数字を入れ換えるプログラム

```
/*  2ツ ノ スウシ゛ ヲ イレカエル　フ゜ロク゛ラム  */

main()
{
        char    c;
①       int     hairetu[2];
/*  ハイレツ ヲ センケ゛ン シマス  */
②       int     *a1, *a2;
/*  ホ゜インタ ヲ センケ゛ン シマス  */
        int     a;
        printf("スウシ゛ ヲ 2ツ ニュウリョク シテク
タ゛サイ=" );注)
③       scanf("%d%c%d", &hairetu[0], &
c, &hairetu[1] );注)
④       printf("イレカエ マエ [0]=%d, [1]=%
d¥n", hairetu[0], hairetu[1] );注)
⑤       a1=&hairetu[0];
⑥       a2=&hairetu[1];
⑦       a=*a1;
⑧       *a1=*a2;
⑨       *a2=a;
/*  ホ゜インタ ヲ ツカッテ ハイレツ ノ テ゛ータ ヲ イレ
カエ マス  */
④       printf("イレカエコ゛ [0]=%d, [1]=%
d¥n", hairetu[0], hairetu[1] );注)
}
```

注）下線部は1行として入力してください。

プログラムの説明

①で、配列を宣言します。配列名の後の、[ ]の中は正の整数で、配列の要素の数を示します。

②で、ポインタ変数を宣言します。ポインタ変数は、変数名の先頭に＊（アスタリスク）をつけて宣言します。

③は、scanf関数で、配列にデータを入力します。

④は、printf関数で、配列を表示します。

⑤～⑨では、ポインタ変数を使って配列のデータを入れ替えます。配列は［添字］によって参照します。添字の値は0（ゼロ）と正の整数です。

ポインタ変数はアドレス（番地）を格納する変数です。ポインタ変数に格納されたアドレスの内容を示します。

配列の宣言

配列の宣言は、配列の型、配列名、配列の要素の数の順に行ないます。

```
int          hairetn[2]  ;
配列の型     配列名 配列の要素の数
```

int型の配列を例に、配列の宣言で確保される領域を図にしてみます。

hairetu
仮に100番地とします

2バイト
2バイト

int型の配列は、int型の変数を、[ ]で指定された要素の数だけ確保します。

配列へのデータの代入

プログラムでは、scanf関数で配列にデータを入力しています。

scanf("%d%c%d", &hairetu[0], &c,&hairetu[1]);

この例では、1 0 , 2 0 EXE と操作します。数字は、「,」(カンマ)で区切って入力します。最初に入力された10がhairetu[0]に、次に入力された20がhairetu[1]に入ります。なお、配列の先頭は[0]から始まります。

データが配列に入力された様子を図にしてみます。

hairetu

hairetu[0]	100	10
hairetu[1]	102	20

hairetu[0]は100番地で内容は10、
hairetu[1]は102番地で内容は20です。

③のscanf関数は、入力する変数名に&をつけます。&はアドレスを表わすポインタ演算子で、scanf関数に変数の先頭アドレスを与えています。

④のprintf関数は、配列を表示します。配列の要素は、添字によって参照します。

配列名[添字]

添字には、式や変数や数字を指定できます。そのときの式の値や数字は0(ゼロ)または正の整数になるようにします。添字は0(ゼロ)から始まり、配列の数－1までです。

コラム●文字列と配列●

文字列を扱うときにも、配列を使用します。
配列名はアドレスを指すので、&はつけません。

```
        :
char mojiretu[10]; /* 10モジ ヲ カクノウ スル モジレツ ノ センゲン */
        :
scanf("%s", mojiretu); /* ハイレツメイ ハ アドレス ヲ シメシマス*/
        :
```


ポインタ変数の宣言と＊演算子

②では、ポインタ変数を宣言しています。

型名　　＊変数名

ポインタ変数は、変数名の先頭に＊（アスタリスク）をつけて宣言します。
プログラム例の、alとa2はint型のポインタ変数として宣言されます。

主なポインタ変数の種類

char　＊変数名;	ポインタ変数をchar型として宣言する
float　＊変数名;	ポインタ変数をfloat型として宣言する
int　＊変数名;	ポインタ変数をint型として宣言する

⑤⑥では、ポインタ変数にアドレスを格納します。この例では、
hairetu[0]は100番地ですから、a1には100が、
hairetu[1]は102番地ですから、a2には102が、
入ります。

＊演算子は、ポインタ変数には格納されたアドレスの内容を示します(＊ポインタ変数)。

⑦⑧⑨では、＊演算子を使って配列のデータの入れ替えを行なっています。
入れ替えには、作業領域として変数aを使用します。
100番地の内容は10ですから、＊a1には10が、
102番地の内容は20ですから、＊a2には20が、
入ります。次の図を参照してください。

アドレス	内　容
100 hairetu[0]	10
102 hairetu[1]	20

アドレス	内　容
a1	100
a2	102

番地の内容	
＊a1	10
＊a2	20

ご注意

◎ポインタや配列を扱うとき、確保した容量を超えてデータの書き込みを行なうと、予期しない動作をしますので、絶対に行なわないでください。

関数定義

ここで学ぶこと
☆関数定義の方法
☆データの受け渡し
☆関数の型宣言

まとめ
◎定義した関数は、main( ) 関数から呼び出して使います。
◎呼び出し元の実引数の値が、呼び出し先の仮引数の値になります。
◎仮引数は呼び出し先で処理されて、結果はreturn文で呼び出し元へ返されます。

〔プログラム例〕 入力された数を２乗して、画面に表示するプログラム

```
/*   ﾆｭｳﾘｮｸ ｻﾚﾀ ｶｽﾞﾖ ｼﾞｼﾞｮｳ ｼﾃ ｶﾞﾒﾝ
ﾆ ﾋｮｳｼﾞ ｽﾙ ﾌﾟﾛｸﾞﾗﾑ */

main ()
{
      int  suu1,suu2;
      printf ("suu1=");
      scanf("%d", &suu1);
      suu2=jijyou(suu1);
①  /* ｶﾝｽｳ ﾆ ﾃﾞｰﾀ ﾖ ﾜﾀｼﾏｽ */
      printf ("suu1*suu1=%d¥n", suu2);
}
②  jijyou (suu3)
③  int  suu3;
{
      int  suu4;
      suu4=suu3*suu3;
④     return (suu4);
/* ｼｮﾘ ﾉ ｹｯｶ ﾖ ｶｴｼﾏｽ */
}
```


プログラムの説明

main( )は、主プログラムを表わす特別な関数でプログラム中に一つだけ書けます。

jijyou( )は、ここで定義している関数です。変数の名前と同じように名前をつけることができます。

①では、②で定義しているjijyou関数を呼び出します。( )の中に変数名を書いて、呼び出す関数にデータを引き渡します。

②では、呼び出されるjijyou関数を定義します。( )の中に書いた変数名で呼び出し元のデータを受け取ります。

③は、呼び出し元から渡されたデータを受け取る変数を宣言しています。

④では、return文により関数の呼び出し元へ戻ります。( )の中に変数名を書くと処理の結果を呼び出し元へ返すことができます。

関数定義の方法

関数は次のように定義します。

```
[型]  関数名 ([引数])
      [引数の型宣言]
      {
            文;
            ⋮
      [return ( );]
      }
```

- [ ]の項目は省略できます。
- [型]を省略するとint型になります。
- 引数を使用しないときは、[引数の型宣言]を省略できます。
- 結果を返さないときは、[return( );] を省略できます。

関数とデータの受け渡し

```
[呼び出し元関数]
    suu2=jijyou(suul);
[呼び出し先関数]
    jijyou(suul);
     {
     }
```

関数の呼び出し元で指定した引数を実引数といいます。実引数の値は呼び出し先の関数の仮引数の値になります。

仮引数は、呼び出す関数名の後で型宣言します。{ }の中で宣言すると仮引数と見なされません(呼び出す関数内で使用する変数宣言になります)。

```
[呼び出し元関数]
    suu2=jijyou(suu1);
[呼び出し先関数]
          return(suu4);
```

return文は、関数の呼び出し元へ戻る文です（プログラム例の④）。関数での処理結果を返すときは、

```
return (式);
        return (suu4);
        return(suu3＊suu3);
```

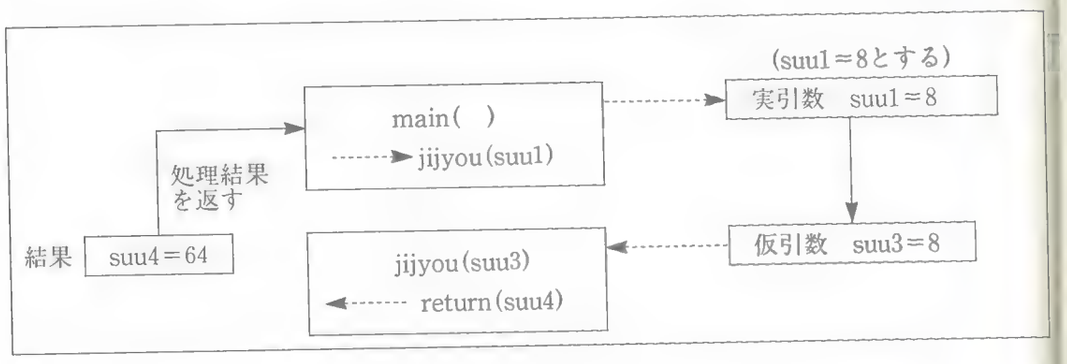
と書きます。

処理結果を返さないときは、

```
return;
```

と書きます。またこの場合は、return文自体を省略できます。

このように引数によってデータを渡し、結果を返す方法をCall by Value（値による呼び出し）といいます。



〔プログラム例〕 入力された数を２乗して１画面に表示するプログラム

```
        /＊ ﾆｭｳﾘｮｸ ｻﾚﾀ ｶｽﾞ ｦ ｼﾞｼﾞｮｳ ｼﾃ ｶﾞﾒﾝ
        ﾆ ﾋｮｳｼﾞ ｽﾙ ﾌﾟﾛｸﾞﾗﾑ ＊/
        main ( )
        /＊ long ｶﾞﾀ ﾉ ﾍﾝｽｳ ｦ ｱﾂｶｴﾙ ﾖｳﾆ ｼﾏｽ ＊/

        {
①           long jijyou();
②           long suu2;
            int suu1;
            scanf("%d", &suu1);
            printf ("%d¥n", suu1);
            suu2=jijyou(suu1);
③           printf ("suu1＊suu1=%ld¥n", suu2);
        }
④       long jijyou(suu3)
        int suu3;
        {
②           long suu4;
```



```
⑤           suu4=suu3;
⑤           suu4=suu4＊suu4;
⑥           return (suu4);
        }
```

long型を扱えるjijyou( )関数を定義しました。

関数の型宣言

①で、呼び出すjijyou関数を、long型に宣言します。

②のように、変数も大きなデータが格納できるようにlong型に宣言します。

③で、％文字はlong型用の％ldを使います。

④で、呼び出されるjijyou関数を、long型で宣言します。

⑤では、long型で計算するため、式の左辺も右辺もlong型にします。

ここではlong型の例をあげて説明をしましたが、float型、double型の関数や変数でCall by Valueするときにはlongを、floatまたはdoubleに書き換えます。

記憶クラス

ここで学ぶこと
☆内部変数と外部変数

文法のまとめ
①外部変数を宣言します。
②内部変数を宣言します。
③実引数でデータを引き渡します。
④仮引数でデータを受け取ります。

〔プログラム例〕 内部変数と外部変数の宣言

```
/* ﾅｲﾌﾞ ﾍﾝｽｳ ﾉ ｾﾝｹﾞﾝ ﾄ ｶﾞｲﾌﾞ ﾍﾝｽｳ
   ﾉ ｾﾝｹﾞﾝ ﾉ ﾚﾝｼｭｳ */
①  int hairetu [2];
/* ｶﾞｲﾌﾞ ﾍﾝｽｳ */
main ()
{
②      static int suu1;
/* ﾅｲﾌﾞ ﾍﾝｽｳ */
       for(suu1=0; suu1 <=1;++suu1){
③          suuset(suu1);
/* ｼﾞｯ ｲﾝｽｳ */
       }
       printf("[0]=%d, [1]=%d", hairetu[0],
       hairetu[1]);注)
}
④ suuset (suu2)
/* ﾊｲﾚﾂ ﾆ ﾃﾞｰﾀ ｦ ｱﾀｴﾏｽ */
int suu2; /* ｶｲﾝｽｳ */
{
⑤      hairetu [suu2]=100+suu2;
}
```

注) 下線部は1行として入力してください。

プログラムの説明

①のように、main( )の前で宣言する変数を外部変数といいます。外部変数は、プログラム中のすべての関数で有効な関数です。

②のように、関数名の後の{ }中で宣言する変数を内部変数といいます。内部変数は他の関数では使用できないので、関数間でデータを受け渡す必要があります。Call by Value でデータを受け渡します。③と④で受け渡しを行なっています。

⑤で使っているhairetu[ ]は、外部変数として宣言してあるので、この例ではreturn 文でデータを返す必要がありません。

内部変数の定義

```
    ⋮
{
    [auto]  (型)  〈変数名1, 変数名2……〉;  …(a)
    static  (型)  〈変数名1, 変数名2……〉;  …(b)
    ⋮
}
```

(a)自動変数

変数を宣言した関数でのみ有効です。他の関数から変数の参照、変更ができないという意味です。

自動変数はメモリーのスタック領域につくられます。他の関数からは変数を参照したり変更したりできません。

なお、autoという記述は省略できます。すなわち何も書かないと自動変数になるということです。また、autoは内部変数にのみ指定できます。外部変数には指定できません。

(b)静的変数

変数を宣言した関数でのみ有効です。他の関数から変数の参照、変更はできません。

静的変数は、メモリーの静的（スタティック）領域につくられます。他の関数からは変数を参照したり変更したりできません。

staticは内部変数にのみ指定できます。外部変数には指定できません。

変数suu1は、内部変数になっていますから宣言した関数でのみ有効です。

外部変数の定義

```
型  変数名1、変数名2……;
main( )
      ⋮
```

外部変数は、プログラム中のすべての関数で有効です。メモリーの静的（スタティック）領域という場所につくられ、すべての関数でこの変数を参照したり変更したりできます。したがって、Call by Valueによるデータの受け渡しは不要です。

コラム●Call by Value●

外部変数はプログラム中のすべての関数で有効な変数ですので一見便利そうですが、乱用すると関数の独立性がそこなわれます。つまり、プログラムミスで気付かないうちに変数のデータを変えてしまうからです。

面倒がらずにCall by Valueを使用して変数を受け渡す方法に慣れたほうが、よいプログラムを書くことができます。

文法のまとめ

ここで学ぶこと

☆文法の復習

まとめ

◎C言語のプログラムは、制御構造になっています。

◎C言語のプログラムは、多くの関数で構成されています。

コメント

プログラム中の、実行に関係ない部分を、コメントといいます。これはプログラムや実行の説明、見出しに書きます。

C言語では、コメントは下の例のように/*と*/で囲まれた部分です。

```
/*    コンニチハ！  ト  シュツリョク  スル  フ゜ロク゛ラム  */
main()
{
        printf("コンニチハ！");
}
```

キーワード

キーワードは、C言語のプログラムの文法で使用するためあらかじめ予約されている言葉です。

キーワードの一覧とその機能を次頁の表に示します。これらのキーワードと同じ名前の変数や関数は宣言できません。また、ここで×印のついているキーワードは、本機のC言語では使用できない機能ですが、キーワードとして登録されていますので変数として使用できません。

表に示すキーワード以外に、ライブラリー関数も変数名として使用できません。

キーワード一覧

	キーワード	意味と用法
	auto	局所変数の記憶クラス指定
	break	for, do, while, switch文からの脱出
×	case	switch文の名札。使用不可
	char	文字型データ(8ビット長)の宣言子
×	const	定数の宣言。使用不可
	continue	for, do, whileにおいて、次の繰り返しへジャンプ
×	default	switch文で該当しないときの飛び先。使用不可
	do	処理の繰り返し実行。do {～} while(式);
	double	倍精度浮動小数点型(64ビット長)の宣言子
	else	if文とともに使用。if(式) {～} else {～}
×	enum	列挙型の宣言子。使用不可
	extern	外部変数や外部定義の記憶クラス指定
	float	単精度浮動小数点型(32ビット長)の宣言子
	for	繰り返し実行。for(式1;式2;式3) {～}
	goto	指定したラベルへのジャンプ
	if	もし式が真ならば実行。if(式) {～}
	int	整数型(16ビット長)の宣言子
	long	倍長整数型(32ビット長)の宣言子
	register	レジスタ変数の記憶クラス指定。autoと同じ
	return	関数の値を返し、呼び出しへ戻る
×	sigened	符号つき型の宣言。使用不可
	sizeof	データ型の長さ。sizeof(型)は使用不可
	short	短整数型の宣言子。intと同じ
	static	静的変数の記憶クラス指定
×	struct	構造体の宣言子。使用不可
×	switch	条件による分岐。使用不可
×	typedef	新しいデータ型の指定。使用不可
×	union	共用体の宣言子。使用不可
	unsigned	符号なし整数型の宣言子
	void	値を返さない関数の型宣言子
×	volatile	プログラムの外側から変更できる型の宣言。使用不可
	while	繰り返しの実行。while(式) {～}

データ型の長さと精度

本機のＣ言語で使用できるデータ型と、その長さを次に示します。

型宣言子	本機のＣ言語
char	8ビット
short	16ビット
int	16ビット
long	32ビット
float	32ビット
double	64ビット

変数名や関数名の付け方

変数名や関数名および＃defineで定義する定数名は、一般に識別子と呼ばれます。識別子には次のような規則があります。

〔識別子の付け方〕

①識別子の１文字目は、英字（A～Z、a～z）またはアンダーバー「_」でなければならない。
また、カナ文字は使えない。

②識別子の２文字目以降は、英字、アンダーバー、数字（０～９）のいずれかでなければならない。また、英大文字と英小文字は区別する。

③識別子に、キーワードおよびライブラリー関数名（次頁の一覧表を参照）を用いてはならない。

④識別子の長さの制限はないが、先頭の８文字までが名前の識別に使われる。つまり、abc012345とabc012344という識別子は同じものとして扱われ、abc012345とabc012335は別のものとして扱われる。

次に識別子として正しいものと正しくないものを示します。

正しい例
var、My_Name、jugemujugemu、V100、XX、count、q1_2_5、int_32 （識別子の一部がキーワードになっているのはよい）

正しくない例	理　由
123、3A　＃count	第１文字目が英字、アンダーバー以外の文字である。
switch、gets	キーワードかライブラリー関数名である。
C-1	英字、アンダーバー、数字以外の文字（ハイフン）がある。

キーワードとライブラリー関数名

abort	default	goto	sinh
abs	do	gotoxy	sizeof
acos	double	if	sprintf
acosh	else	inport	sqrt
angle	enum	int	sscanf
asin	exit	log	static
asinh	exp	log10	strcat
atan	extern	long	strchr
atanh	fflush	main	strcmp
auto	fgetc	malloc	strcpy
beep	fgets	outport	strlen
break	float	pow	struct
breakpt	for	printf	switch
calloc	fprintf	putc	tan
case	fputc	putchar	tanh
char	fputs	puts	typedef
clearerr	free	register	union
clrscr	fscanf	return	unsigned
const	getc	scanf	void
continue	getch	short	volatile
cos	getchar	signed	while
cosh	gets	sin	

注）上記のほか、内部システムで使用している関数名があります。これらの名前も識別子として使用することはできません。その場合、エラーとなりますので、他の名前を使用してください。

ご注意

◎Ｃ言語では、通常名前のつづりは小文字を用いますが、＃defineで検知する記号定数（EOF、BUFFSIZE、…）には大文字を使用して、混合を避けるようにしています。VX-4のＣ言語もこの記入方法に従っています。

定数データとデータ表現

文字定数

char型の定数は「 ' 」(シングルクォーテーション) で文字を囲むことによって表わします。

'C'　　'8'　　'¥n'

VX-4では、文字コードとしてASCIIコードを使用しています。

上の例の '¥n' は「エスケープシーケンス」と呼ばれるものであり、英字や数字などで表現できない文字を表現するためのものです。

ASCIIコードを例に示します。詳しくは操作テキストの巻末の「キャラクターコード一覧表」を参照してください。

キャラクター	ASCII コード(10進)	ASCII コード(16進)
A	65	41
Z	90	5A
0	48	30
9	57	39

キャラクターコード一覧表の中のBS、CRなどは、制御文字(Control Character)で、特別な意味を持つ文字定数です。C言語では、これらの制御文字の代わりにエスケープシーケンスが用意されています。それらを次に示します。

表記	呼び名(略名)	意　味
¥a	Bell(BEL)	音を鳴らす
¥n	New Line(NL)	改行 (+復帰)
¥t	Horizontal Tablation(HT)	水平タブ
¥b	Back Space(BS)	バックスペース (1文字後退)
¥r	Carriage Return(CR)	復帰 (行の先頭へ移動)
¥f	Form Feed	改ページ
¥¥		¥という文字
¥'		'という文字
¥"		"という文字
¥0	Null	00Hのコード、値0
¥nnn	8進表示	nnnに8進数を記入するとその文字コードの意味になる。
¥xdd	16進表示	ddに16進数を記入するとその文字コードの意味になる。

例を次に示します。これらは文字定数(1バイトの定数)です。

(例)　'¥n'　'¥b'　'¥¥'　'¥033'

整数定数

int型の定数を表記する方法は大きく分けて10進、16進、8進の3種類があります。long型の定数は、最後にLまたは1をつけます。

整数定数

10進表示	例	1121　−128　6800　123456789L(longの場合)
	注意	後述の8進数と区別するため、0で始まってはならない。ただし、0だけの場合はよい。
16進表示	例	0x1B　0xFFFF　0x09　0xffffffL(longの場合)
	注意	0xまたは0Xを先頭につける。
8進表示	例	033　0777　012　01234567L(longの場合)
	注意	0を先頭につける。

本機で扱う整数型定数は次の通りです。負の定数は、符号なし定数に単項マイナス演算子がついたものです。

	型	例
10進定数	int long	0〜32767 32768〜2147483647
8進定数	int unsigned int long	00〜077777 0100000〜0177777 0200000〜017777777777
16進定数	int unsigned int long	0x0000〜0x7FFF 0x8000〜0xFFFF 0x10000〜0x7FFFFFFF

上記範囲以外は不定となります。また、scanf、fscanf、sscanfによる入力も上記範囲となります。

浮動小数点定数、倍精度浮動小数点数

小数点を含むfloat型、double型の定数は次のような形式で記述します。指数部はEまたはeを用いて記述します。

(例)　3.1416　−1.4142　1.0e−4　1.23456E5

本機で扱えるfloat型、double型定数の範囲は次の通りです。

float	0, ±1e−63 〜 ±9.99999e+63
doudle	0, ±1e−99 〜 ±9.9999999999e+99

文字列定数

これは、文字型定数を並べたような形式で、「"」(ダブルクォーテーション) によって文字の並びを囲みます。文字列の構造は、文字列の最後に「¥0」(NULL) がつけられます。例を次に示します。

"How do you do"

演算子

C言語には、BASIC、FORTRANやPascalなど他の言語には見られない多くの演算子があります。

演算子は、＋、－、＊、／などのように、変数などの値を変化させる処理を行ないます。

数多くあるC言語の演算子には、優先順位と結合規則というものが決められています。

演算子の種類と機能

一次演算子	(～)、func(～)	カッコ、関数の引数の演算
	x〔～〕、y〔～〕〔～〕	配列要素の指定
単項演算子	＊px	ポインタの示す内容の指定
	&x	変数xのアドレス
	＋x 、－x	xの正値または負値
	＋＋x、－－x	変数xを使う前に＋1または－1
	x＋＋、x－－	変数xを使った後で＋1または－1
	～x	xのビットごとのnot(反転)
	!x	xの論理not(xが0以外なら0、0なら1)
	(型)x	xの強制型変換・指定(cast演算)
	sizeof(x)	変数xのバイト長の値。sizeof(型)は使用不可
二項演算子	x＊y、x／y、x％y	乗算、除算、剰余(xをyで割った余り)
	x＋y、x－y	加算、減算
	x≪y、x≫y	xをy回、左ビットシフト、右ビットシフト
	x＜y、x＜＝y、x＞＝y	xとyの大小比較(真ならば1、偽ならば0)
	x＝＝y	等値比較(等しければ1、等しくなければ0)
	x!＝y	不等値比較(等しくなければ1、等しければ0)
	x&y	xとyのビットand
	x^y	xとyのビット排他的or(xor)
	x¦y	xとyのビットor
	x&&y	xとyの論理的and(xとyが共に0以外なら1)
	x¦¦y	xとyの論理的or(xとyのどちらかが0以外なら1)

代入演算子	x＝y x＊＝y ↔ x＝x＊y x／＝y ↔ x＝x＊y	変数xにyを代入 xにyをかけてxに代入 xをyで割ってxに代入
	x％＝y、x＋＝y、x－＝y、x≪＝y、x≫＝y、x&＝y、x¦＝y、x^＝y (説明省略)	
順序演算子	x、y (a＝b＋c、x＝a/2 ;)	x、yを評価して結果としてyを与える。 例(aにd＋cを代入し、つぎにxにa/2を代入。 このxの値を与える)

つぎに、以上示した演算子の種類と優先順位それに結合規則を表で示します。

演算子の種類と優先順位および結合規則

優先順位	演算子		結合規則
高い		( ) 〔 〕	→
	単項	! ^ ＋＋ －－ (型) ＊注1) & sizeof	←
	乗除	＊注2) ／ ％	→
	加減	＋ －	→
	シフト	≪ ≫	→
	比較	＞ ＜ ＞＝ ＜＝	→
	等価比較	＝＝ !＝	→
	ビットAND	&.	→
	ビットXOR	^	→
	ビットOR	¦	→
	論理AND	&&	→
	論理OR	¦¦	→
	代入	＝ ＋＝ －＝ ＊＝ ／＝ その他	←
低い	順序	,	→

→ 左から右に演算
← 右から左に演算

注1) 間接指定記号の ＊
注2) 乗算記号の ＊

制御構造

C言語のプログラムは、実行の単位である文をいくつも書き、それらを、順次実行させるという構成になっています。

制御構造とは、文の実行の

①順序を変える
②繰り返す
③条件によって分岐させる

ことです。

文

文の最小単位は、データや演算子や関数呼び出しを含んだ単一の式を実行させる文です。式のあとに;を付けます。

普通、文は次のような代入か関数呼び出しです。

```
x = sin(a + b) - c;
printf( "x=%10.4lf¥n" , x) ;
a = b+ (c=getchar() ) ;
```

複文(ブロック)

複文は、いくつかの文を { と } でくくって、1つの文にまとめたものです。文法的には1つの文と同じとして取り扱われます。終わりを示す } の後ろには普通は ; はつけません。

```
{
        文1;
        文2;
        ……
        文n;
}
```

複文

```
main ( )
{ int i;
  {  int  j;
  }
}
```

上のような宣言並びでの指定はできません。

分岐や繰り返しなどの制御構造

条件分岐	
if (式) 　文;	①式が真ならば、 ②文を実行します。

if (式) {
　文;
}
else{
　文;
}
	①式が真ならば、
	②文1を実行し、
	
	③それ以外ならば(偽ならば)、
	④文2を実行します。

if (式1) {
　文1;
}
else if (式2) {
　文2;
}
else if (式3) {
　文3;
……
}
else if (式n)
　文n;
}
else {
　文n+1;
}
	①式1が真ならば、
	②文1を実行し、
	
	③それ以外で(偽で)、式2が真ならば、
	④文2を実行し、
	
	⑤それ以外で(偽で)、式3が真ならば、
	⑥文3を実行し、
	……
	
	⑦それ以外で(偽で)、式nが真ならば、
	⑧文nを実行し、
	
	⑨それ以外なら(式1～式nが偽なら)
	⑩文n+1を実行します。

繰り返し制御	

while (式) {
　文;
}
	①式が真のあいだ
	②文を実行します。

for (式1 ; 式2 ; 式3) {
　文;
}
	①式1を実行して初期設定を行ないます。
	②式2が真なら、文を実行します。
	③式3を実行して条件を更新をします。
	④式2が真なら、文を実行します。
	⑤以下、式2が真のあいだ、文の実行、式3による更新を繰り返します。

do { 　文; } while(式) ;	①文を実行します。 ②式が真なら、元に戻ります。 ③以下、式が真のあいだ、文の実行を繰り返します。

無限ループ	
for(;;) 　文 ;	①文を繰り返し実行します。 （for文の初期設定、条件判断、条件更新がない）
while(1) 　文 ;	①文を繰り返し実行します。 （while文の条件がいつも真（1））
do 文; while(1) ;	①文を繰り返し実行します。 （do～while文の条件がいつも真（1））
break文、continue文	
while（1）｛ …… if（式）break ; …… ｝	①while文による無限ループが実行されます。 ②式が真ならbreak文が実行され、無限ループから抜け出します。
while（式1）｛ …… if（式2）continue ; …… ｝	①式1が真のあいだ、｛　｝の中が実行されます。 ②式2が真なら、continue文が実行され、while文に戻り、式1が実行されます。
無条件ジャンプ	
goto ラベル ; …… ラベル ;	①ラベルの位置に無条件にジャンプします。

記憶クラス

語憶クラスは、宣言した変数を確保するメモリー領域を指定したり、変数の通用範囲（プログラムのどの範囲から読み書きできるか）を指定するものです。

記憶クラスには次の種類があり、それぞれ変数の使い方によって使い分けます。

記憶クラス	変数の使い方
自動変数　　auto （内部変数）	宣言した関数内でだけ有効。その範囲を出たら、無効となります。一時的な変数。 変数はstackエリアに確保されます。 autoの記述は省略してもよい。
静的変数　　static （内部変数）	宣言した関数内でだけ有効。 変数はsymbolエリアに確保されます。 外部変数にこの指定を行なうとエラーになります。
レジスタ変数　　register （内部変数）	VX－4では、autoと同じ意味として扱われます。
外部変数　　extern （外部変数）	プログラム中のどこでも宣言できる外部変数。 プログラムエリア内のどこかにあるexternなしの外部変数を参照するために使います。 外部変数はどの関数でも参照できます。 変数はsymbolエリアに確保されます。

注意

①auto変数は、次のプログラム(a)のように、関数に入った直後の先頭で宣言する内部変数です。プログラム(b)のように、関数内の実行の途中で変数の宣言をするとエラーになります。

auto変数の例

プログラム(a) OK

```
func(a)
double a ;
{
  int   i, j, x;
  float fx, fy ;
  long lx, ly ;
  ……
{
```

auto変数は、関数の先頭で宣言します。

プログラム(b)エラー

```
func(a)
double a;
{
  int i, j, x ;
  printf("%lf", a) ;
  float fx, fy;
    ……
{
```

② 下の、(a)のような外部変数は宣言できません。(b)または(c)のように宣言します。プログラム(b)とプログラム(c)では、xの有効範囲が異なることに注意してください。

static変数の例

プログラム(a)エラー

```
static int x[5];
main()
{
      .....
}
```

プログラム(b) OK

```
int x[5];
main()
{
      .....
}
```

プログラム(c) OK

```
main(  )
{
      static int x[5];
      .....
}
```


配列とポインタ

配列

VX−4のC言語では、n次元配列まで使用することができます。

(例)　int m[2][12][13];

配列の初期化

VX−4のC言語では、配列の初期化はできません。配列の初期化には、次のように代入演算を行なってください。

```
char *month[12];
month[0]="Jan";month[1]="Feb"; ····· ; Month[11]="Dec";
```

ポインタ

ポインタは変数のアドレスをもつ変数で16ビット長です。

int x, *px; x=*px;	左の例ではpxが指している内容がxに代入されます。

int x,y, *px;
px=&x;
y=*px;
	左の例ではxのアドレスがpxに代入された後、pxが示すアドレスの内容をyに代入します。
	すなわち、y=x;と同じことになります。

main()
{
char *p;
p="Casio";
printf("%c %s ¥n", *p,p);
}
	文字列においてポインタを用いると、文字列を単一文字に分解することができます。実行させると結果は次のようになります。
	C Casio
	ポインタpはCasioのCを指します。

main()
{
char *p;
p="Casio";
printf("%c %c ¥n", *p, *(p+2));
}
	CasioのCとsを取り出すには左のようにします。

main()
{
int i,a[5], *pa;
pa=a;
for(i=0;i<=4;i++)
*(pa+i)=i;
for(i=0;i<=4;i++)
printf("%d" a[i]);
printf("¥n");
}
	配列の各要素はポインタで示すこともできます。左の例では配列aの各要素をpaで示しています。
	配列名aは、配列の最初の要素a[0]を指すポインタを表わします。したがって、配列の各要素を指すポインタは次のようになります。
	pa ····· a[0]
	pa+1 ····· a[1]
	pa+2 ····· a[2]
	pa+3 ····· a[3]
	pa+4 ····· a[4]

```
main()
{
    char *pc, c[80];
    pc = c;
    strcpy (pc, "abcdefgh");
    printf("%s¥n" ,pc);
}
```

ポインタは変数のアドレスを持っているだけです。したがって、場合によっては格納領域の確保は別に行なう必要があります。

左の例では、ポインタpcと80文字の格納領域を確保するために配列cを宣言し、ポインタpcを配列のポインタcと共通にしています。

ポインタを用いると明快で単純なプログラムを作ることができる反面、予想しないアドレスを指すポインタを作ることもありますので、注意して用いてください。

また、ポインタや配列を扱うとき、確保した容量を超えてデータの書き込みを行なうと、予期しない動作をしますので、絶対に行なわないでください。

関数

一つの仕事を行なう単位が「関数」です。C言語のプログラムは、多数の関数から構成されるのが普通です。Cのプログラムを作るということは関数を作るということとほぼ同じ意味です。

関数には、あらかじめ用意されているprintfやscanf、getcharなどの入出力関数、sinやconなどの数学関数および文字列を処理する関数などがあります。これらを「ライブラリー関数」と呼びます。ライブラリー関数のそれぞれについては、リファレンスの「ライブラリー関数」で詳しく説明します。

ユーザーが自分自身で新しい関数を書く場合の注意点を説明します。

main( )関数

main( )関数は主プログラムであることを示す特別な関数で、プログラムの一番はじめに起動します。

右のような、プログラムでコマンド入力ラインを調べることはできません。プログラム中でscanf関数などを使ってキー入力を行なうようにしてください。

```
main(argc, argv)
int argc;
char *argv[ ];
{
    ...............
}
```

関数の型宣言

int型以外の値を返すには、関数の型宣言が必要です。例として、double型の値を返す関数dsquareを作って、1から10までの2乗を出力するプログラムを示します。

```
main()
{
      double d, dsquare();
      for(d=1.0;d<=10.0;d+=1.0)
            printf("(%lf)^2=%lf¥n", d
, dsquare(d)); 注)←呼び出し先
}                 ↑      仮引数
               実引数      ↓
double dsquare (x)     ←呼び出し先
double x;  ←仮引数はここで宣言します
{
      return(x*x);   ←関数での処理の結果を返します
}
```

注) 下線部は1行として入力してください。

このような方法をCall by Valueといいます。

このように、int以外を値として返す関数では、関数の宣言ばかりでなく、使う側(main関数内)でも、宣言が必要です。

また、次に示すように、記憶クラスのextern宣言をして使用することができます。

```
extern double dsquare ();
main()
{
    double d;
  for(d=1.0;d<=10.0;d+=1.0)
        printf("(%lf)^2=%lf¥n", d
, dsquare(d));注)
}
double dsquare (x)
double x;
{
    return(x*x);
}
```

注）下線部は1行として入力してください。

その他の注意点として、次のようなことに留意してください。

- 再帰呼び出し：本機のC言語においても、再帰呼び出しを使用することができます。
- void関数宣言が可能です。
- 次のような、関数へのポインタの宣言はできません。

```
main()
{
  void pr(), (*pr_p) ();
  pr_p=pr;
  pr ();
  (*pr_p) ();
}
void pr()
{
  printf("In pr_func¥n");
}
```

④ライブラリー関数は型指定宣言せずに使用できます。一般のCコンパイラでは、stdio.hやmath.h、ctype.hなどのヘッダファイルを取り込まなければ、ライブラリー関数が使えない場合がありますが、VX−4ではこのようなヘッダファイルを取り込まなくてもライブラリー関数を使用できます。

プリプロセッサ

C言語の処理系には、実行部分をコンパイルする前に、プリプロセッサと呼ばれる前処理を行なう機能が備わっています。VX−4 のC言語では、次のように記述してマクロ展開（#define）が使用可能です。

```
#define  識別子  文字列  （定数の定義や文字列の置換）
                 └─（省略できません）
```

また、次の点にご注意ください。

①VX−4では、#define文によって次のような引数付きマクロ定義をすることできません。

```
#define ispper(x) (x>'a')
```

………使用できません。

代わりに、次のような関数を作ってください。

```
isupper(x)
int x;
{
    return(x>='A' && x<='Z'
}
```

………代わりに使用してください。

②VX−4では次の情報があらかじめ登録されています。

```
#define NULL 0
#define EOF -1
#define FILE char
FILE *stdin;
FILE *stdout;
FILE *stderr;
FILE *stdprn;
```

ただしstdin、stdout、stderr、stdprnを使用する場合は、extern宣言を行なう必要があります。

マニュアルコマンドリファレンス

RUN

（書式） RUN
RUN>"PRN："
RUN>"prn："

（機能） プログラムをロードして実行させます。
"PRN："または"prn：" を指定すると、実行結果をプリンタに出力します。
指定を省略すると、画面に出力します。

EDIT

（書式） EDIT

（機能） プログラムの編集を行なうエディタに入ります。
プログラムの実行でエラーが発生したときにEDITを入力すると、エディタに入りエラー発生行を表示します。
ブレークでプログラムの実行が中断しているときにEDITを入力すると、エディタに入りブレークした行を表示します。

TRON

（書式） TRON

（機能） トレースしながらプログラムを実行するトレース機能を指定します。トレース機能を指定してプログラムを実行すると、プログラムを１行実行するごとに、次に実行する行を表示します。
トレース中のキー操作　　EXE キー：実行を続行します。
C キー：実行を続行します。
BRK キー：実行を中断します。
トレース機能は、起動時はOFFになっています。

TROFF

（書式） TROFF

（機能） トレース機能を解除します。

ライブラリー関数リファレンス

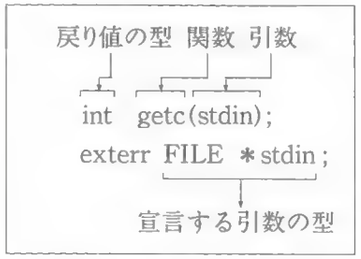
ライブラリー関数

あらかじめ定義されているので、ユーザーが定義しないで呼び出して使える関数が、ライブラリー関数です。
本機のC言語には、次ページ以降に示す入出力関数、プロセス関数、メモリー関数、文字列関数、数値関数などが用意されています。

ライブラリー関数リファレンスの見かた

リファレンスでは、書式、戻り値の型、機能、例を示します。

（例）



「書式」については、次のように示してあります。

戻り値の型	関数によって得られる戻り値の型を示します。 戻り値の型は、次の型宣言子で記述します。 char short int long float double void
関　　数	関数名です。関数の次の「(」と「)」の中に引数を書きます。 「;」は、文の区切りを意味する記号です。
引　　数	引数を書きます。 引数が２つ以上のときは、「,」(カンマ)で区切ります。 引数のない関数では、「( )」のようにカッコだけを書きます。
宣言する引数の型	関数で使用する引数の型を関数名の後に(下に)指定します。 引数のない関数では何も指定しません。

入出力関数

getchar

（書式） int getchar();　（戻り値） 読み込んだ文字のコード。int型。

（機能）
キーボードから1文字読み込みます。
getcharは、getc(stdin)と同じです。
入力は、EXEキーを押すと行なわれます。
SHIFT EXEと押すと、EOF(−1)が入力されます。

（例）
```
int c;
    ⋮
c=getchar();
```

getc

（書式） int getc(stdin);
extern FILE *stdin;
（戻り値） 読み込んだ文字のコード。int型。

（機能）
読み込んだ1文字のコードを返します。
入力は、EXEキーを押すと行なわれます。
動作は、getcharと同じです。
stdin(キーボード)以外からの入力を指定することはできません。

（例）
```
extern FILE *stdin;
int c;
    ⋮
c=getc(stdin);
```

fgetc

（書式） int fgetc(stdin);
extern FILE *stdin;
（戻り値） 読み込んだ文字のコード。int型。

（機能）
キーボードから1文字読み込みます。
入力は、EXEキーを押すと行なわれます。
動作はgetcharと同じです。
stdin(キーボード)以外からの入力は指定できません。

（例）
```
extern FILE *stdin;
int c;
    ⋮
c=fgetc(stdin);
```

putchar

（書式） int putchar(c);
int c;
（戻り値） 出力した文字のコード。int型。

（機能）
stdout(表示画面に)1文字出力します。

（例）
```
main()
{
  char buffer[64];
  int i, c;
  gets(buffer);
  for(i=0; buffer[i]!='¥0';i++){
      c=putchar(buffer[i]);
      if(c==EOF)break;
  }
}
```


putc

（書式） int putc(c, stdout);
int putc(c, stdprn);
int c;
extern FILE *stdout, *stdprn;
（戻り値） 出力した文字のコード。int型。

（機能）
stdout(表示画面)またはstdprn(プリンタ)に1文字出力します。

fputc

（書式） int fputc(c, stdout);
int fputc(c, stdprn);
int c;
extern FILE *stdout, *stdprn;
（戻り値） 出力した文字のコード。int型。

（機能）
stdout(表示画面)またはstdprn(プリンタ)に1文字出力します。
動作は、putcと同じです。

（例）
```
extern FILE *stdout;
char buffer[30];
int c;
    ⋮
c=fputc(buffer[0], stdout);
```

gets

（書式） char *gets(string);
char *string;
（戻り値） 格納されたデータのポインタ。char型。

（機能）
stdin(キーボード)から1行読み込み、それをstringに格納します。
文字列は、改行文字まで読み込まれます。
改行文字は、string中では「¥0」(NULL)文字に置き換えられます。

（例）
```
char string[30], *result;
    ⋮
result=gets(string);
```

fgets

（書式）
```
char *fgets(string, count, stdin);
char *string;
int count;
extern FILE *stdin;
```

（戻り値） 格納されたデータのポインタ。char型。

（機能）

stdin（キーボード）から文字列を読み、それをstringに格納します。

文字は、改行文字または読み込んだ文字数がcount-1になるまで読まれます。文字列の最後に「¥0」(NULL)文字が付加されます。

改行文字が読まれた場合は、string中で「¥0」(NULL)文字に置き換えられます。

（例）
```
entern FILE *stdin;
char string[30], *result;
        :
result=fgets(string, 30, stdin);
```

puts

（書式）
```
int puts(string);
char *string;
```

（戻り値） 改行文字コード。int型。

（機能）

stdout（表示画面）にstringを出力します。

文字列の終了を表わす「¥0」(NULL)文字は、改行文字に置き換えて書き込みます。

（例）
```
int result;
        :
result=puts("string");
```

fputs

（書式）
```
int fputs(string, stdout);
int fputs(string, stdprn);
char *string;
extern FILE *stdout, *stdprn;
```

（戻り値） 書き込んだ最後の文字。int型。

（機能）

stdout（表示画面）またはstdprn（プリンタ）にstringを出力します。

文字列の終了を表わす「¥0」(NULL)文字を書き込みません。

（例）
```
extern FILE *stdprn;
int result;
        :
result=fputs("string", stdprn);
```


printf
fprintf
sprintf

（書式）
```
printf (format[, argument……]);
int fprintf (stdout, format[, argument……]);
int fprintf (stdprn, format[, argument……]);
int sprintf (buffer, format[, argument……]);
char *format;
char *buffer;
extern FILE *stdout, *stdprn;
```

（戻り値） 出力した文字数。int型。（sprintfでの最後の「¥0」(NULL)文字は数えません）エラーなら、EOF。

（機能）

argumentをformatにしたがって変換し、printfはstdout（表示画面）に、fprintfはstdout（表示画面）またはstdprn（プリンタ）に、sprintfはbufferに、それぞれ出力します。

sprintfの場合だけ、最後に「¥0」(NULL)文字を出力します。

formatは、0個以上の文字列で、普通の文字、エスケープシーケンス、変換仕様からなります。普通の文字とエスケープシーケンスは、現われる順にそのまま出力されます。

argumentが変換仕様よりも多いときは、余分なargumentは無視され、少ないときは結果が不定となります。

（例1）
```
int count;
count=234;
sprintf("%d%06d%X%x%o¥n",
   count, count, count, count, count);
```
出力結果　234 000234EA ea 352

（例2）
```
int count;
count=234;
printf("|%d|%6d|%-6d ¥n",
   count, count, count);
```
出力結果　|234|　　234|234　　|

書式変換仕様

変換仕様は、「%」ではじまる次の文字列です。

%[フラグ][フィールド幅][.精度][l]変換文字

ただし、「%」の後に変換仕様として意味をもたない文字が続く場合には、その文字列以降、次の「%」まではそのまま出力されます。

また、「%」を出力するには、「%%」とします。

変換仕様の文字列

フラグ「－」

変換結果をフィールド内で左寄せにします。デフォルト値は「右寄せ」です。

フィールド幅

最小フィールド幅を符号なしの10進整数で指定します。

変換結果がフィールド幅より小さい場合は左側はパッド文字で埋められます。ただし、フラグに「－」が指定してあるときには、右側がパッド文字で埋められます。

パッド文字は、フィールド幅を指定する10進整数が「0」ではじまっている場合は「0」、それ以外はスペースとなります。

変換結果がフィールド幅より大きい場合、またはフィールド幅が指定されていない場合は、変換結果がそのまま出力されます。

精度

精度は、ピリオドを前に付けた符号なしの10進整数で指定します。ピリオドのみが書かれ、10進整数が省略されたときは「0」として扱われます。変換文字により、次のような意味をもちます。

「d, o, u, x, X」　出力する最小の桁数を指定します。引数の桁が精度よりも少ない場合、出力値の左側は「0」で埋められます。デフォルトは「1」です。

「e, E, f」　小数点の後に出力する桁数を指定します。精度が「0」の場合、小数点も出力されません。デフォルトは「6」です。

「g, G」　出力する最大の有効桁数を指定します。デフォルトは、「すべての有効桁数」です。

「s」　出力する最大の文字数を指定します。デフォルトは、「¥0(NULL) 文字が出て来るまで」です。

「その他」　その他の文字が書かれると、精度は無視されます。

l

変換文字により、次のような意味をもちます。

「d, o, x, X」　引数がlong型であることを指定します。

「その他」　その他の文字が書かれると、「l」は無視されます。

変換文字

「d」　整数引数が符号付10進表記に変換されます。

「o」　符号なし8進表記に変換されます。

「u」　符号なし10進表記に変換されます。

「x, X」　符号なし16進表記に変換されます。

「f」　double引数が、[－]ddd. dddの形の10進小数点表記に変換されます。(dddは1つ以上の10進数) ここで、小数以下の桁は四捨五入され精度指定に等しくなります。

「e, E」　double引数が、[－]d. ddde±dd(eのとき)または[－]d. dddE±dd(Eのとき)の形に変換されます(dは1つ、dddは1つ以上、ddは2つの10進数)。ここで、小数以下の桁は四捨五入され精度指定に等しくなります。

「g, G」　double引数が、fまたはe(GならばE)の形に変換されます。精度指定は、有効数字桁数を示します。e(またはE)が選択されるのは、値の指数が－4未満または精度指定よりも大きいときです。後方の0は、結果から除かれます。小数点の文字は、後ろに数字が続くときだけに付きます。

「c」　整数引数を1文字として出力されます。

「s」　引数は文字列へのポインタで、文字列の終端の「¥0」(NULL)文字まで(含まない)、または精度で指定された文字数に達するまでが出力されます。

注) 変換文字 luはuと見なされます。

注) argumentが浮動小数点以外で、「f」、「e,E」、「g,G」の浮動小数点を表示する変数文字を使用すると、「＃NAN」と出力される場合があります。

scanf
fscanf
sscanf

（書式）

```
int scanf (format [, argument……]);
int fscanf (stdin, format [, argument……]);
int sscanf (buffer, format [, argument……]);
char *format;
char *buffer;
extern FILE *stdin;
```

（戻り値） 代入されたargumentの個数(0もあり得ます)。int型。

（機能）

入力したデータをformatにしたがって変換し、argumentに代入します。

scanfとfscanfはstdin(キーボード)から、sscanfはbufferから入力します。

argumentは、formatで指定された型に対応する型の変数を指すポインタです。

scanfとfscanfは、EXEキーを押すと入力されます。sscanfは、「¥0」(NULL)文字がbufferの終わりと見なされます。

formatは、0個以上の文字列で、空白文字、普通文字、変換仕様からなります。変換仕様が現れると、仕様にしたがって変換された値が後続のargumentに順番に代入されます。

argumentが変換仕様よりも多いときは、余分なargumentは無視されます。少ないときは、結果が不定となります。

（例）

```
int i;
float f;
doubled;
acanf ("%d%f%1f", & i, & f, & d);
```

1 2 3 EXE − 1 · 2 3 E
1 0 EXE 2 0 3 と入力すると、iに123、fに−1.23e10、dに203.0が代入されます。

書式変換仕様

変換仕様は、「%」ではじまる次の文字列です。

```
%[*][フィールド幅][l]変換文字
```

変換仕様は、次のステップの順に実行されます。

①入力された空白文字はスキップされます。ただし、変換文字が「c」のときは除きます。

②入力フィールドが読み込まれます。入力フィールドは、最初の空白文字まで、または変換仕様で変換できない文字まで、または指定されていればフィールド幅までのどれかが最初に起きるまでです。入力フィールドの次の文字は、まだ読まれていないものとみなされます。もし、入力フィールドの長さが「0」ならば、この関数は終了します。

③入力フィールドは、変換文字により定まる型に変換されます。

④代入禁止「*」が指定されていると、入力フィールドは読み込まれますが、変換結果は「&」で示すメモリーに代入されません。

変換仕様の文字列

＊(代入禁止)

入力フィールドは読み込まれますが、変換結果のargumentへの代入は行なわれません。

フィールド幅

最大フィールド幅を符号なしの10進整数で指定します。

l

変換文字により、次のような意味を持ちます。

「d, o, x, X」	引数がlong型であることを指定します。
「e, f, g」	引数がdouble型であることを指定します。
「その他」	その他の文字が書かれると、「l」は無視されます。

変換文字

「d」	10進整数（符号が付いていてもよい）とマッチします。 対応する引数は、整数へのポインタであること。
「o」	8進整数（符号が付いていてもよい）とマッチします。 対応する件数は、整数へのポインタであること。
「u」	符号なしの10進整数とマッチします。 対応する引数は、整数へのポインタであること。
「x」	16進整数（符号が付いていてもよい）とマッチします。 対応する引数は、整数へのポインタであること。
「e, f, g,」	浮動小数点数（符号が付いていてもよい）とマッチします。 対応する引数は、floatへのポインタであること。 指数の数字列の入力が3桁以上の時は結果が不定となります。
「s」	非空白文字からなるシーケンスにマッチします。 対応する引数は、このシーケンスと終端の「¥0」(NULL)文字を収容するのに十分な大きさを持つ文字配列へのポインタであること。 終端の「¥0」(NULL)文字は、自動的に付加されます。
「c」	フィールド幅で指定された個数の文字からなるシーケンスにマッチします。 (フィールド幅指定がない場合は、1とみなされます。) 対応する引数は、この文字列を収容するのに十分な大きさを持つ文字配列へのポインタであること。空白文字は、スキップされないで読み込まれます。

注）変換文字 luはuと見なされます。

fflush

（書式） int fflush(stdin);
extern FILE *stdin;

（戻り値） 正常ならば、0。int型。

（機能）
バッファの内容をクリアーします。

（例）
```
extern FILE *stdin;
int c;
  ⋮
c=getc(stdin);
fflush(stdin);
```

getch

（書式） int getch( );

（戻り値） 読み込んだ1文字のコード。int型。

（機能）
stdin(キーボード)から直接1文字読み込み、そのキャラクターコードを返します。読み込んだ値は表示されません。
入力バッファが空の場合は、カーソルが点滅して待機します。

（例）
```
int x;
  ⋮
x=getch();
```

inport

（書式） int inport(n);
int n;

（戻り値） 読み込んだ値。int型。

（機能）
nで指定されたポートから1バイト読み込みます。
nは、0から7の範囲です。

outport

（書式） void outport(n, i);
int n, i;

（戻り値） 何も返しません。

（機能）
nで指定されたポートにiを出力します。
nは0から7の範囲、iは0～255の範囲です。

clearerr

（書式） void clearerr(stdin);
extern FILE *stdin;

（戻り値） 何も返しません。

（機能）
入力バッファ内のEOFをクリアーします。

プロセス関数

breakpt

（書式） void breakpt( );

（戻り値） 何も返しません。

（機能）
プログラムの実行を停止し、ブレークモードに入ります。

（例）
```
  ⋮
breakpt( );
  ⋮
```

コラム●ブレークモード●

breakpt関数が実行されたとき、またはトレース中にBRKキーが押されると、ブレークモードに入り「Break?」というメッセージが表示されます。
ブレークモードでは次のようなキー操作を行ないます。

キー	機能
A	実行を終了
C	実行を再開
EXE	実行を再開
T	トレースしながら実行を再開
N	トレースしないで実行を再開
D	変数名を入力すると変数の型とブレークしたときの変数の値を表示 (もう一度BRKキーを押すと変数表示から抜け出す)

exit

（書式） void exit( );

（戻り値） 何も返しません。

（機能）
プログラムを正常終了させます。正常終了させる前に出力バッファの内容をクリアーします。

abort

（書式） void abort( );

（戻り値） 何も返しません。

（機能）
プログラムを異常終了させます。このとき、「Abort」というメッセージをstdout(表示画面)に出力します。

メモリー関数

malloc

（書式） char ＊malloc (size);
unsigned size;

（戻り値） 確保されたメモリー領域のポインタ。char型。確保できなかった場合は「¥0」(NULL)を返します。

（機能）
sizeで指定した大きさのメモリー領域を確保します(単位はバイトです。)
確保されたメモリー領域は、プログラムの実行終了とともに解放されます。

（例）
```
main ( )
{
char ＊c ;
    ⋮
if ((c＝malloc(256))＝＝NULL){
  printf("データリョウイキガトレマセン¥n");
  exit ( );
  }
    ⋮
}
```

calloc

（書式） char ＊calloc (n, size);
unsigned n;
unsigned size;

（戻り値） 確保したメモリー領域のポインタ。char型。確保できなかった場合は「¥0」(NULL)を返します。

（機能）
sizeで指定した大きさ(バイト)のn個の要素の配列をメモリー領域に確保し、0で初期化します。
確保されたメモリー領域は、プログラムの実行終了とともに解放されます。

（例）
```
main ( )
{
int ＊iarry, i;
    ⋮
it ((iarry＝(int＊)calloc(1000,2))＝＝
  NULL){
  printf("ハイレツガトレマセン¥n");
  exit ( );
  }
for (i＝0; i<1000;i＋＋) iarry[i]＝0;
    ⋮
}
```


free

（書式） int free (ptr);
char ＊ptr;

（戻り値） 解放されると0。int型。ptrが無効だと(calloc、mallocによって確保されたメモリー領域のポインタでないと)、−1を返します。

（機能）
mallocやcallocで確保されたメモリー領域を解放します。
ptrで、calloc、mallocによって確保されたメモリー領域のポインタを指定します。

（例）
```
char ＊arry;
    ⋮
arry＝malloc(256);
    ⋮
free (arry);
```

文字列関数

strlen

（書式） int strlen (string);
char ＊string;

（戻り値） 文字列stringの「¥0」(NULL)を含まない長さ。int型。

（機能）
文字列stringの「¥0」(NULL)文字の直前までのバイト数を返します。

（例）
```
int length;
    ⋮
length＝strlen ("adc");
                /＊length＝3＊/
```

strcpy

（書式）
char ＊strcpy (dest, source);
char ＊dest, ＊source;

（戻り値） destのポインタ。char型。

（機能）
文字列sourceの先頭から「¥0」(NULL)文字まで(「¥0」(NULL)を含む)の範囲を、文字列destの後ろにコピーします。
コピーするとき、オーバーフローのチェックは行ないません。

（例）
```
char ＊result, string [64];
    ⋮
result＝strcpy (string, "abc");
                /＊string＝"abc"＊/
```

strcat

（書式）
char strcat (dest, source);
char ＊dest, ＊source;

（戻り値） destのポインタ。
char型。

（機能）
文字列sourceの先頭から「¥0」(NULL)文字の直前までの範囲を文字列destの最初の「¥0」(NULL)文字の後ろに付加し、最後に「¥0」(NULL)文字を付加します。
付加のとき、オーバーフローチェックは行ないません。

strcmp

（書式）
int strcmp (string1, string2);
char ＊string1, ＊string2;

（戻り値） 次の整数値を返します。
int型。
string1＜string2→0より小さい整数
string1＝string2→0
string1＞string2→0より大きい整数

（機能）
文字列string1と文字列string2の先頭から一文字ずつ、「¥0」(NULL)文字が現われるまで比較(ASCIIコード順)します。
「¥0」(NULL)も比較の対象となります。

（例）

```
int result;
char string1[5];
char string2[5];
    ⋮
strcpy (string1, "abcde");
strcpy (string2, "abcda");
result＝strcmp (string1, string2 );
                        /＊result＝1＊/
```


strchr

（書式）
char ＊strchr (string, chr);
char ＊string;
int chr;

（戻り値）
検索したchr(指定文字)のポインタ。char型。
見つからなかったときは「¥0」(NULL)を返します。

（機能）
文字列stringで最初に現われる指定文字chrを検索します。
「¥0」(NULL)文字も検索の対象となります。

（例）

```
main ( )
{
  char instr[64], ＊ss;
  printf ("Input string＝");
  gets (instr);
  ss＝instr;
  while ((ss＝strchr (ss,'＊'))!＝NULL)
                    /＊アスタリスク ノ
                      ケンサク＊/
    ＊ss＝'_';   /＊アンダーバー ニ
                      オキカエ＊/
  puts (instr);
}
```

数値関数

abs

（書式） int abs (n);
　　int n;

（戻り値） nの絶対値。int型。

（機能）
整数の絶対値を返します。

（例）
```
main ( )
{
   int i, ans;
   i=−1;
      ⋮
   ans=abs (i);
      ⋮
}
```

sin / cos / tan

（書式） double sin (x);
　　double cos (x);
　　double tan (x);
　　double x;

（戻り値） double型。

（機能）
角度xに対する三角関数の値を返します。
角度xが演算範囲を超えている場合は、エラーになります。
角度xの演算範囲は、次の通りです。
|x| ＜1440 (DEG)
|x| ＜8π　(RAD)
|x| ＜1600 (GRA)

（例）
```
main ( )
{
   double y;
   angle (0);
   for (;;) {
      printf ("Angle?");
      scanf ("%lf",&y);
      printf ("%11.10g%11.10g%11.10g
¥n", sin(y), cos(y), tan(y));
   }
}
```


asin / acos / atan

（書式） double asin (x);
　　double acos (x);
　　double atan (x);
　　double x;

（戻り値） double型。

（機能）
xに対する逆三角関数の値（角度）を返します。
xが演算範囲を超えている場合には、エラーになります。
xの演算範囲は次の通りです。
　−1≦x≦1(asin, acosの場合)
　|x| ＜100^{100} (atanの場合)
返される関数の値の範囲は次の通りです。(RADの場合)。
　[−π/2, π/2](asin)
　[0, π](acos)
　[−π/2, π/2](atan)

（例）
```
double y;
   ⋮
angle (1);
y=asin (1.0);
y=acos(1.0);
y=atan(1.0);
```

sinh / cosh / tanh

（書式） double sinh (x);
　　double cosh (x);
　　double tanh (x);
　　double x;

（戻り値） double型。

（機能）
角度xに対する双曲線関数の値を返します。
　sinh x ＝$(e^x-e^{-x})/2$
　cosh x ＝$(e^x+e^{-x})/2$
　tanh x ＝$(e^x-e^{-x})/(e^x+e^{-x})$
xが演算範囲を超えている場合は、エラーになります。
xの演算範囲は、次の通りです。
　|x| ≦230.2585092(sinh, coshhの場合)
　|x| ＜100^{100}, −1≦|tan(x)|＜1
(tanhの場合)

（例）
```
double y;
   ⋮
angle (0);
y=sinh(1.0);
y=cosh(1.0);
y=tanh(1.0);
```

asinh / acosh / atanh

（書式） double asinh(x);
double acosh(x);
double atanh (x);
double x;

（戻り値） double型。

（機能）
xに対する逆双曲線関数の値を返します。
$\sinh^{-1}x=\log_e(x+\sqrt{x^2+1})$
$\cosh^{-1}x=\log_e(x+\sqrt{x^2-1})$
$\tanh^{-1}x=\log_e(1+x/1-x)/2$
xが演算範囲を超えている場合は、エラーになります。
xの演算範囲は次の通りです。
$|x|<5E+99$ (asinh)
$1\leqq x<5E+99$ (acosh)
$-1<x<1$ (atanh)

（例）
```
double y;
    ⋮
angle(0);
y=asinh(1.0);
y=acosh(2.0);
y=atanh(0.5);
```

pow

（書式） double pow(x,y);
double x, y;

（戻り値） double型。

（機能）
xのy乗(X^Y)の値を返します。
yが0の場合は、1を返します。
xが0でyが負の場合と、xが負でyが整数でない場合は、エラーになります。
結果がオーバーフローの場合も、エラーになります。

（例）
```
double x, y, z;
x=2.0; y=3.0;
    ⋮
z=pow(x, y);
```

sqrt

（書式） double sqrt(x);
double x;

（戻り値） double型。

（機能）
xの平方根($\sqrt{x}$)を返します。
xが負の場合は、エラーになります。

（例）
```
double y;
    ⋮
y=sqrt(2.0);
```


exp

（書式） double exp(x);
double x;

（戻り値） double型。

（機能）
xの指数関数(e^x)の値を返します。
x＞230.2585092の場合、エラーになります。

（例）
```
double y;
    ⋮
y=exp(1.0);
```

log / log10

（書式） double log(x);
double log10(x);
double x;

（戻り値） double型。

（機能）
logは、xの自然対数($\log_e x$)の値を返します。
log10は、xの常用対数($\log_{10} x$)の値を返します。
xが0の場合、負の場合はエラーになります。

（例）
```
double y;
    ⋮
y=log(1000.0);
y=log10(1000.0);
```

angle

（書式） void angle(n);
unsigned n;

（戻り値） 何も返しません。

（機能）
三角関数、逆三角関数の角度モードを指定します。nの指定により次のようになります。
0:DEG （度）
1:RAD （ラジアン）
2:GRA （グラッド）

（例）
```
double y;
    ⋮
angle(0);
y=sin(90.0);
angle(1);
y=sin(1.570796327);
angle(2);
y=sin(100.0);
```

その他の関数

beep　（書式）void beep(n);
unsigned n;　（戻り値）何も返しません。

（機能）
nの指定により、低い音か高い音を鳴らします。
0:低い音を鳴らします。
1:高い音を鳴らします

（例）
```
beep(0);
```

clrscr　（書式）void clrscr();　（戻り値）何も返しません。

（機能）
表示画面の表示を消去し、カーソルをホームポジションに戻します。

（例）
```
clrscr();
```

gotoxy　（書式）void gotoxy(x, y);
unsigned x;
unsigned y;　（戻り値）何も返しません。

（機能）
仮想スクリーン(32桁×8行)上のカーソル位置を、xとyで指定します。
座標は、仮想スクリーン上の左上隅を原点(0,0)とし、次の範囲で指定できます。

$0 \leqq x \leqq 31$
$0 \leqq y \leqq 7$

（例）
```
gotoxy(10, 2);
```


第2章 BASIC編

プログラムの構成

ここで学ぶこと
☆BASICプログラムの構成

まとめ
◎BASICプログラムは、入力文、出力文、演算文で構成されます。
◎ひとつの文は、行番号、コマンド、オペランドで構成されます。
◎プログラムの文には、実行される文と、実行されない文があります。

プログラム例

```
10 REM "REI コンニチハ ノ プログラム"
20 PRINT "コンニチハ"
30 END
```

プログラムフローチャート

プログラムを作るためには、いろいろな文法や規則があります。文法などの詳しい説明は後にして、まずBASICプログラムがどのような形になっているのかを見てみましょう。

プログラムの基本

実行される命令文では、基本的に3つの文に分けられます。

(1)データを入力するための入力文
(2)そのデータに基づいて演算などの処理をする演算文
(3)実行の結果を出力するための出力文

すなわち、データを入力し、そのデータを演算などの処理で加工し、加工された実行結果を出力する……という一連の作業が、プログラムの基本になるわけです。

文(行)の基本構成

BASICプログラムの各文(行)は、基本的な約束ごとにしたがって構成されています。

プログラムを書いたり実行したりするためには、各文(行)に名前を付けておく必要があります。BASICでは、各文(行)に番号を付けて、それを名前とします。これが「行番号」です。

行番号は1～65535までの整数で、小さい順に付けていきますが、習慣として10、20、30のように10きざみで付けるようにしましょう。

行番号に続いて計算機に命令する「コマンド」、そのコマンドに必要な情報を示す「オペランド」を記述します。

実行文と非実行文

BASICプログラムの文には、入出力や演算などの処理をする実行文と、処理をしない非実行文とがあります。プログラム例の行番号10のREM文は非実行文のひとつで、注釈(コメント)の役割をしています。

インタープリターの操作

ここで学ぶこと
☆プログラムを入力するための準備

まとめ
◎本機でBASICを使用する前に、使用するプログラムエリアを指定します。
◎使用するプログラムエリアに、すでにプログラムが入力されている場合、NEWコマンドでプログラムを消去します。

プログラムエリア

これからプログラムを入力したり実行したりするための準備として、使用するプログラムエリアを指定します。

ONキーを押して本機の電源を入れます。

```
-
```

MODE 1と操作すると、次のような表示になります。

```
P 0 1 2 3 4 5 6 7 8 9        3355B
Ready P0
-
```

これでBASICモードになり、プログラムの作成・編集・実行ができます。

表示の1行目の「P」はプログラムエリアを意味し、「0」から「9」までの数字は各プログラムエリアを示しています。「3355B」は、残りのメモリー容量を示しています。(RAMパックを増設している場合や、データバンク、C言語、CASLのプログラムを記憶している場合は、この数値が異なります。)

2行目の「Ready P0」は、本機が現在「P0」エリアを使用していることを示しています。使用するプログラムエリアを変更するには、SHIFTを押した後に、「0」から「9」までの数字キーを押し、指定することができます。

プログラムの消去

1行目の「0」から「9」までの数字の代わりに「＊」が表示されている場合は、そのプログラムエリアにプログラムがすでに入力されています。もし使用したいプログラムエリアが「＊」になっている場合は、「NEW」を指定しプログラムを消去しましょう。

右の表示は、プログラムエリア0、1にすでに入力されているときのものです。

```
P * * 2 3 4 5 6 7 8 9        3044B
Ready P1
-
```


ここでNEWEXEのキーを押すと、Readyで示されたプログラムエリア(右の表示ではP1)の内容が消去されます。

```
P * 1 2 3 4 5 6 7 8 9        3108B
Ready P1
-
```

またNEWALLEXEと操作すると、すべてのプログラムエリアの内容が消去されます。

プログラムの入力と実行

ここで学ぶこと
☆BASICプログラムの入力方法
☆入力されたプログラムを確認する方法
☆プログラムの実行方法

まとめ
◎プログラムエリアを指定してから、BASICプログラムを入力します。
◎EDITコマンドやLISTコマンドで、入力したプログラムを確認します。
◎RUNコマンドで、入力済みのプログラムを実行します。

簡単なプログラムを使って、プログラムの入力・編集・実行をする方法を説明します。ここではプログラムで使用している命令については考えず、操作方法だけをマスターしてください。

プログラムの入力

前述の「プログラムの構成」で示したプログラムを、プログラムエリア0に入力してみましょう。

操作		表示	
①	1 0 SPC R E M SPC SHIFT " _ R E I SPC SHIFT カナ K O N N I T I H A SPC N O SPC P U R O G U R A M U SHIFT " SHIFT カナ EXE	①	10 REM "REI コンニチハ ノ フ゜ロク゜ラム"
②	2 0 SPC SHIFT PRINT SHIFT " _ SHIFT カナ K O N N I T I H A SHIFT " SHIFT カナ EXE	②	20 PRINT "コンニチハ"
③	3 0 SPC E N D EXE	③	30 END

プログラムの入力が終了したら、BRKキーを押し、入力作業を終了します。

入力上の注意

行番号20のように、SHIFT PRINT_と押すと、PRINTという文字列を入力することができます。このようにして入力することを「ワンキーコマンドを使う」といいます。ワンキーコマンドを使用すると、入力されたコマンドの後ろに自動的に1個のスペースがつきます。

アルファベットは、BASICの基本として大文字で入力してください。大文字と小文字のモードの切り換えは、CAPSキーを押します。本機のBASICでは、コマンドが小文字で入力されても大文字に変換されます。ただし、変数名や「"」(ダブルクォーテーション)で囲まれた文字列は、入力されたままになります。

プログラムの確認

入力されたプログラムを確かめるには、次の2つの方法があります。

(1)LISTコマンドを使用する……プログラムを見る

L I S T EXEと操作すると、画面に次々とプログラムが表示されます。

```
20 PRINT "コンニチハ"
30 END
Ready P0
_
```

しかし、これでは速すぎてプログラムを読むことができません。そこで↓キーや↑キーを使って、現在表示されている行以外の行を確認します。

また、LISTコマンドで長いプログラムを表示中にSTOPキーを押すと、プログラムリストの表示を一時停止させることができます。これは少しずつ見るときに便利です。次の部分を表示させるにはEXEキーを押します。

本機は8行分の仮想スクリーンを持っているので、現在表示されている行を含めて8行を↑↓キーで確認することができます。

L I S T 行番号 ……指定した行番号を表示します。
L I S T 行番号 − 行番号 ….指定した範囲の行番号を表示します。

(2)EDITコマンドを使用する……プログラムを編集する

SHIFT EDIT_ EXE と入力すると、画面にプログラムが表示されます。この状態で、EXE(前進)、SHIFT EXE(後進)、↑、↓キーを使って、他の行を確認することができます。

```
10 REM "REI コンニチハ ノ フ゜ロク゜ラム"
20 PRINT "コンニチハ"
30 END
```

プログラムの確認が終了したら、BRKキーを押します。「Ready P0」が表示され、作業は終了します。

プログラムリストを確認中に、プログラムの入力ミスを発見した場合は、プログラムの修正を行ないます。↑↓←→キーを使って、カーソル(画面上で点滅している「_」)を修正したい部分に移動させ、修正後EXEキーを押します。このとき、文字と文字の間に挿入したいときはINSキーを押して間を空けます。逆に文字を消去して、後ろの文字を前に詰めたいときは、SHIFT DEL_を押します。修正方法については次ページの「エディタの操作」を参照してください。

抜けている行を発見したときには、行番号とコマンド、オペランドを入力し、最後にEXEキーを押すと行の挿入ができます。不要な行を発見したときには、行番号を入力後に、EXEキーを押します。入力した行番号の文が1行分が削除されます。

プログラムの実行

「Ready P0」が表示されているときに、SHIFT RUN_ EXEを押すとプログラムエリア0に入力されているプログラムが実行されます。

```
RUN
コンニチハ
Ready P0
_
```

エディタの操作

ここで学ぶこと
☆プログラムの修正方法
☆実行エラーの対処方法
☆プログラムのデバッグ方法

まとめ
◎プログラムの入力中またはLISTコマンドやEDITコマンドで確認中は、いずれもカーソルを移動させて修正します。
◎実行中のエラーは、LISTコマンドまたはEDITコマンドでプログラムを表示させてから修正します。
◎エラーメッセージの意味を理解します。
◎STOPコマンドやトレース機能でプログラムの誤りを発見します。

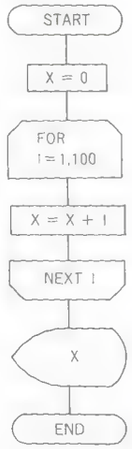
プログラムの修正を行なうのは、次の４つの場合です。
(1)プログラム入力中(EXEキーはまだ押していない)にミスに気付いた。
(2)リスト確認中(EXEキーを押してしまった後)にミスに気付いた。
(3)実行してみたらエラーが出た。
(4)エラーはないが、実行結果がおかしい。

このようにいろいろな場合がありますが、文の訂正、挿入、削除の方法は同じです。いずれの場合も、修正したいところにカーソル(画面で点滅している「_」)を移動させて修正します。
プログラムリストの修正はBASICモード(MODE 1)で行ないます。

プログラム列

```
10  X=0
20  FOR  I=1  TO  100
30  X=X+I
40  NEXT  I
50  PRINT  X
60  END
```

上に示したプログラム例を使って、いろいろな修正方法を紹介します。このプログラムは1から100までの整数の和を計算するものです。プログラムの内容の理解は後回しにして、いろいろな修正方法をしっかりと覚えてください。なおプログラムエリア１に「入力中」または「入力後」という設定で修正を行なうものとします。



プログラムフローチャート

入力中のプログラム訂正(EXEキーを押す前)

プログラムの入力中(EXEキーはまだ押していない)にミスに気付いたときは、カーソルを訂正部分に移動させ、修正します。

(例１) 30 X＝X＋I を30 X＝Y＋I と入力してしまった。

```
20 FOR I=1 TO 100
30 X=Y+I_
```

←←←
(カーソルを訂正部分に移動させます。)

```
20 FOR I=1 TO 100
30 X=Y+I
```

X EXE
(正しい文字を入力し、EXEキーを押すとこの行が登録されます。)

```
20 FOR I=1 TO 100
30 X=X+I
_
```

入力後のプログラム訂正(EXEキーを押した後)

LISTコマンドで訂正する方法

プログラムリストを、入力後(EXEキーを押した後)にミスに気付いたときは、その行を表示させ、カーソルを訂正したい部分に移動させ、訂正します。

(例２) 30 X＝X＋I を30 X＝XX＋I と入力してしまったことに気付いた。

L I S T 3 0 EXE
(訂正したい行を表示します。)

```
30 X=XX+I
Ready P1
_
```

↑↑→→→→→
(カーソルを、訂正したい文字まで移動させます。)

```
30 X=XX+I
Ready P1
```

SHIFT DEL EXE
(余分な文字を削除します。)

```
30 X=X+I
Ready P1
```

ここで、正しく訂正されたか確認してみましょう。

CLS
(表示されている文字をクリアーします。)

```
_
```

L I S T 3 0 EXE
(訂正された行を表示します。)

```
30 X=X+I
Ready P1
_
```

正しく訂正されたことが確認できます。

EDITコマンドで訂正する方法

プログラムを、EDITコマンドで表示させ⇧⇩キーを使って修正したい行を画面の最上行に表示させます。次に⇦⇨キーで訂正したい部分にカーソルを移動させます。

（例３） 30 X＝X＋Iを30･X＝＋Iと入力してしまったことに気付いた。

SHIFT EDIT EXE

```
10 X=0
20 FOR I=1 TO 100
30 X=+I
40 NEXT I
```

⇩⇩
（訂正したい行を1行目に表示させます。）

```
30 X=+I
40 NEXT I
50 PRINT X
60 END
```

⇨⇨⇨⇨⇨⇨
（カーソルを訂正したい部分に移動させます。）

```
30 X=+I
```

INS
（1つスペースを空けます。）

```
30 X=_+I
```

X EXE
（正しい文字を入力します。）

```
30 X=X+I
40 NEXT I
50 PRINT X
60 END
```

もし他にも訂正する箇所があれば、このまま⇧⇩⇦⇨を使って訂正したい行を画面の最上行に表示させ、カーソルを訂正させたい位置まで移動して同様に訂正できます。

ご注意

◎LISTコマンドでは、仮想スクリーン８行の中だけで訂正を行ないますが、EDITコマンドでは、プログラム全体を見ながら訂正できます。

EDITコマンドでは、「まず⇧⇩を使って訂正させたい行を１行目に表示させ、次に⇦⇨を使ってカーソルを移動させる」ということに注意してください。

実行してエラーが出たときの訂正

実行時のエラーにはいろいろな種類があります。プログラムに不都合な点があると、RUNコマンド（SHIFT RUN EXE）を入力した後、次の例のようにエラーメッセージが表示されます。

エラーメッセージ　エラーが発生したプログラムエリア
↓
SN error P1-20
↑
この部分の略号で　エラーが発生した行番号
エラーの内容がわ
かります。

この例では、「SN error」すなわち「コマンドまたは文の書式に誤りがある」ということになります。（エラーメッセージの内容は、操作テキストの巻末の「エラーメッセージ一覧表」を参照してください。）LISTコマンドやEDITコマンドで間違いのある行を表示し、訂正します。訂正方法は前に説明した方法と同じです。

コラム●プログラムのデバッグ●

プログラムの誤りを「バグ」といいます。バグとは虫（BUG）のことで、この虫を取り除く作業のことを「デバッグ（DEBUG）」といいます。

本機では自動チェック機構を利用し、プログラムの流れや、BASICプログラムの文法上の誤り（記述ミス）がないかを、表示を見ながらデバッグ作業を行なえます。

デバッグ作業には、「エラーメッセージによるデバッグ」「STOPコマンドによるデバッグ」「トレースモードによるデバッグ」の方法があります。

●エラーメッセージによるデバッグ

キー入力やプログラムに誤りがあったり、プログラムの実行中になんらかの不都合が発生すると、エラーメッセージとエラーの発生した場所を表示して実行が中断されます。

ここで、L I S T ・ EXE またはE D I T ・ EXE とキーを押すとエラーの発生した行が表示されます。プログラムをよく検討し、訂正します。

●STOPコマンドによるデバッグ

STOP文をプログラム中に書き込んでおくと、そこでプログラムの実行が停止されます。調べたい変数などのある行の直後にSTOP文を書き込み、実行させるとそこで停止しますので、

SHIFT PRINT 調べたい変数名 EXE と操作して変数の内容を表示させます。

実行を再開させるには、SHIFT CONT EXE と操作します。

●トレースによるデバッグ

トレース機能をONにすると、行番号を表示しながらプログラムを実行することができます。トレースONの指定は、

T R O N EXE と操作します。トレース機能は、プログラムの流れが思い通りにいっているかどうかを「追跡」するのに用いると便利です。トレースONの解除は、

T R O F F EXE と操作します。

その他、エラー表示が出力されずにプログラムは実行するが、実行結果が期待と違う場合は、原因として、

・引用している変数の間違い
・配列の引数の間違い
・指定すべき分岐先の間違い

などが考えられます。注意深くプログラムを確認し、修正し再実行してください。

条件分岐するプログラム

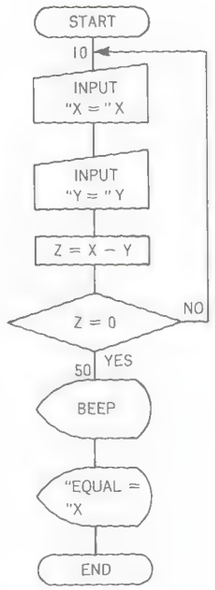
ここで学ぶこと
☆データの入力、表示
☆変数名のつけかた
☆IF文による条件式判断

まとめ
◎INPUT文はデータ入力をします。
◎LET文は代入の指定をします。
◎IF文は条件判断により異なった実行をさせることができます。
◎BEEP文はブザーを鳴らします。
◎PRINT文はデータを表示します。
◎END文はプログラムを終了します。

プログラム例

```
①  10  INPUT "X=";X
①  20  INPUT "Y=";Y
②  30  LET Z=X-Y
③  40  IF Z=0 THEN 50 ELSE 10
④  50  BEEP
⑤  60  PRINT "EQUAL=";X
⑥  70  END
```

(XとYの2つの数値を入力し、XとYの値が等しければブザーを鳴らしEQUAL＝とXの値とを表示し、XとYの値が等しくなければ再度入力することを求めるプログラム。)



プログラムフローチャート

プログラムの文法と要点

①INPUT文

キーボードからデータを指定した変数に入力します。

```
表示される文字列   入力される変数名
        ↓          ↓
INPUT  "X=";  X
```

②LET文

左辺の変数に右辺の値，文字または式の値を代入します。

```
省略可能  代入される変数名
  ↓        ↓
 LET      Z=X-Y
```

LETコマンドは、「LET」の記述を省略できます。

変数とは数値や文字列を貯えておく「箱」のようなものです。変数の名前を変数名といいます。変数名のつけかたには次のような規則があります。

(1)先頭の１文字がアルファベット(Ａ～Ｚ、ａ～ｚ)またはカナ(ア～ン)を使い、２文字目からは数字も使うことができます。

(2)変数名の先頭に予約語(BASICの命令や関数)を含むことはできません。

(例) PRINTER, LETTER

(3)変数名の最後に＄記号をつけることにより文字変数となります。

(4)変数名の長さは最大255文字です。

変数は意味のある変数名をつけるとよいでしょう。

(例) TEIHEN、TAKASA、GOUKEI

③IF～THEN～ELSE文

IF以下の条件式が成立したとき、THEN以下の文を実行、または指定された行番号へ分岐します。条件式が成立しなかったとき、ELSE以下の文を実行、または指定された行番号へ分岐します。

```
        条件式   成立しなかったときに分岐する行番号
          ↓                        ↓
IF  Z=0   THEN  50  ELSE  10
                  ↑
          成立したときに分岐する行番号
```

条件式には次の記号が使用できます。

記号	例	意　味
＝	A＝B	AとBが等しい
＜	A＜B	AがBより小さい
＞	A＞B	AがBより大きい
＜＝ ＝＜	A＜＝B A＝＜B	AがBより小さいか等しい
＞＝ ＝＞	A＞＝B A＝＞B	AがBより大きいか等しい
＜＞ ＞＜	A＜＞B A＞＜B	AとBが等しくない

④BEEP文

ブザーを鳴らします。BEEPまたはBEEP 0 で低い音を鳴らし、BEEP1で高い音を鳴らすことができます。

⑤PRINT文

文字列、数式、変数などの出力要素を表示します。

```
    表示する文字列   表示する変数
          ↓          ↓
PRINT  "EQUAL=" ; X
```

⑥END文

プログラムの実行を終了し、コマンドの入力待ちとなります。

コラム●省略可能なIF文、LET文、END文●

- IF～THEN～ELSE文のなかで、条件式が成立しなかった場合に実行する処理がなかったときは、ELSE以下を省略することができます。
- LET文では「LET」を省略し、計算式や代入文だけを記述できます。
- END文自体を省略することができます。プログラムは実行する文がなくなった時点で自動的に終了します。

繰り返しのプログラム

ここで学ぶこと

☆特定の文の実行を、ある回数だけ繰り返す方法

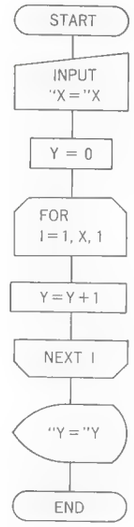
まとめ

◎FOR～NEXT文は制御変数値を変化しながら、ある回数だけ繰り返して実行させることができます。

プログラム例

```
① 10  INPUT "X=";X
② 20  Y=0
③ 30  FOR I=1 TO X STEP 1
④ 40  Y=Y+I
③ 50  NEXT I
⑤ 60  PRINT "Y=";Y
⑥ 70  END
```

（1から、入力したXの値までの和を求めるプログラム。）



プログラムフローチャート

プログラムの文法と要点

①INPUT文

キーボードからデータを指定した変数に入力します。

②代入文(LET文)

左辺の変数に右辺の値，文字または式の値を代入します。「LET」は省略してあります。行番号20では、合計値の入る変数Yを0にしています。

③FOR～NEXT文

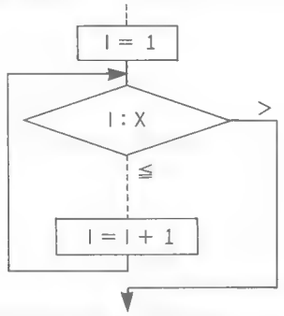
FOR文からNEXT文までの間に書かれた文を、制御変数を初期値から終値まで増分値ずつ変化させながら繰り返します。

```
制御変数名   初期値  終値        増分値
     ↓       ↓     ↓           ↓
FOR  I=    1  TO  X  STEP  1
     繰り返し実行される文
NEXT  I ←この時点でIの値が増分される
```

例では行番号30～50の文に、行番号40の文がはさまれています。この行番号30～50までをFOR～NEXTループと呼び、行番号40の文が繰り返し実行する文になります。

このように、FORとNEXTは常にペアで使用されます。また、初期値、終値、増分値は変数でも数値でも指定することができます。それでは図を見ながらもう一度FOR～NEXTの動きを確認してみることにします。

説明図　この図は、FOR～NEXTの動きを説明するための図で、プログラムとは一致していません。



……制御変数に初期値をセットする

……制御変数と終値を比較し、制御変数の値が終値を超えたときに、NEXT文の次の文にジャンプします

……制御変数に増分値を加え、FOR文の終値との比較へ戻る

図のように繰り返し実行される文は、制御変数と終値とを比較した後に実行されるため、初期値が終値を超えた場合、一度も実行されません。

また、FOR～NEXTループはメモリーが許す限り何重にも指定することもできます。その場合、他のループに重なってはいけません。

```
FOR  J=1  TO  10          FOR  J=1  TO  10
  FOR  I=1  TO  10          FOR  I=1  TO  10
  NEXT  I                   NEXT  J
NEXT  J                   NEXT  I
```

正しい例(内側より実行される)　　誤った例

④代入文(LET文)

左辺の変数に右辺の値，文字または式の値を代入します。「LET」は省略してあります。行番号40では、合計値の入る変数YにIを加算しています。

⑤PRINT文

文字列、数式、変数などの出力要素を表示します。

⑥END文

プログラムの実行を終了し、コマンドの入力待ちとなります。

コラム●省略可能な増分値●

FOR～NEXT文の中のSTEPの記述を省略することができます。STEP以下を省略した場合、増分値は1が指定されたものとみなされます。

無条件ジャンプのプログラム

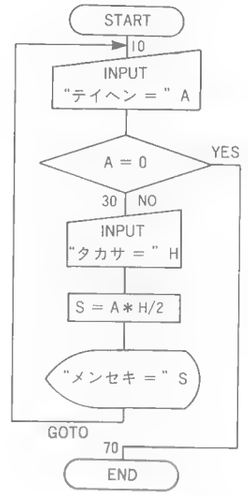
ここで学ぶこと
☆プログラムの流れを変え、特定の行番号へジャンプする方法

まとめ
◎乗算(掛け算)は＊で表わし、除算(割り算)は／で表わします。
◎GOTO文は指定された行番号へ無条件にジャンプします。

プログラム例

```
①  10  INPUT "テイヘン=";A
②  20  IF A=0 THEN 70
①  30  INPUT "タカサ =";H
③  40  S=A*H/2
④  50  PRINT "メンセキ=";S
⑤  60  GOTO 10
⑥  70  END
```

(三角形の底辺と高さを入力してその面積を求めそれを表示し、底辺が0の場合は終了するプログラム。)



プログラムフローチャート

プログラムの文法と要点

①INPUT文

キーボードからデータを指定した変数に入力します。

②IF文

IF以下の条件式が成立すると、THEN以下の行(行番号70)へ分岐します。
成立しないと次の行(行番号30)へ進みます。

③代入文(LET文)

左辺の変数に右辺の値，文字または式の値を代入します。「LET」は省略してあります。
算術式には次のような記号があり、次の優先順で計算されます。

優先順位	種類	記号
1	カッコ	( )
2	乗算　除算	＊　／
3	加算　減算	＋　－

注) 優先順位が同じ場合は左から順に計算されます。

④PRINT文

文字列、数式、変数などの出力要素を表示します。

⑤GOTO文

指定された行番号へ無条件でジャンプします。

```
        ジャンプする行番号
            ↓
GOTO   10
```

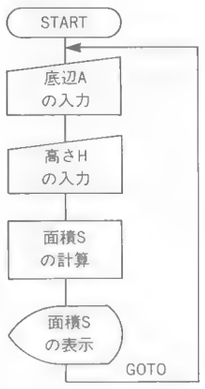
この例では、行番号10にジャンプします。

これまでに学んだプログラムでは、IF～THEN～ELSE文やFOR～NEXT文のような文を除き、行番号順に実行され、最後の文を実行するとプログラムの実行が終了しました。このようなプログラムでは、同じ処理を何回も繰り返して実行したいとき、もう一度はじめからRUNコマンドを操作しなければなりません。このような面倒な操作をしなくても、GOTO文の指定により特定の行にジャンプして繰り返し処理することができます。

⑥END文

プログラムの実行を終了し、コマンドの入力待ちとなります。

コラム●無限ループ●



例題のプログラムでは、Aの入力値が0のときに処理が終了するように設定されていますが、行番号20のIF文がないと左のフローチャートのような流れになります。

このフローチャートには終了がありません。無限に数値の入力と計算、表示を行なうことができるかわりにいつまでたってもプログラムは終了しません。

このように終わりがなく、無限に繰り返される繰り返しを、「無限ループ」といいます。本機で実行を止めるにはBREAKキーを押して、強制的に無限ループから抜け出さなければなりません。

無限ループの存在するプログラムは、あまり好ましいプログラムとはいえません。どうしても無限ループとなってしまう場合は、無限ループのどこかで脱出する方法を考えておく必要があります。

コラム●FOR～NEXTループ内からの脱出●

FOR～NEXTループ内から、GOTO文でループ外へジャンプすることができます。この場合、NEXT文が現われる以前の行にGOTO文などで戻すことにより、再びFOR～NEXTを続行することができます。

また、これとは逆に、FOR～NEXTループ外からループ内に入ることはできません。

サブルーチンの使い方

ここで学ぶこと

☆同じ処理を1つにまとめ、サブプログラム化する方法

まとめ

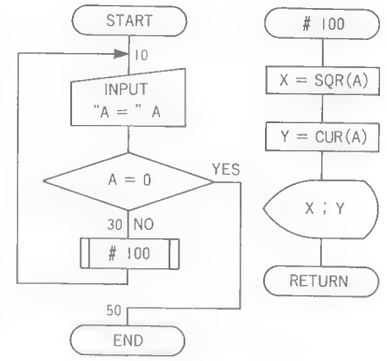
◎GOSUB文は、メインプログラム（メインルーチン）よりサブプログラム（サブルーチン）へ実行を移します。

◎RETURN文は、サブプログラムへ実行を移したメインプログラムのGOSUB文の次の文へ実行を戻します。

プログラム例

```
①   10  INPUT "A=";A
②   20  IF A=0 THEN 50
③   30  GOSUB 100
⑦   40  GOTO 10
⑧   50  END
④  100  X=SQR(A)
④  110  Y=CUR(A)
⑤  120  PRINT X;Y
⑥  130  RETURN
```

（入力したAの値の平方根Xと立方根Yを求め、それを表示し、Aの値が0の場合は終了するプログラム）



プログラムフローチャート

プログラムの文法と要点

①INPUT文

キーボードからデータを指定した変数に入力します。

②IF文

IF以下の条件式が成立するとTHEN以下の行（行番号50）へジャンプします。成立しないと次の行（行番号30）へ進みます。

③GOSUB文

指定された行番号から始まるサブルーチンへジャンプします。サブルーチンからはRETURN文で戻ります。

```
        ジャンプ先の行番号
            ↓
GOSUB   100
```

④代入文(LET文)

左辺の変数に右辺の値・文字または式の値を代入します。「LET」は省略してあります。

行番号100では変数Aの平方根を、行番号110では変数Aの立方根をそれぞれ計算し変数X、Yへ代入しています。本機には内蔵関数とそのための専用キーが用意されています。

内蔵関数には数値関数の他に、文字列の変換などの文字関数などがあります。詳しくは、操作テキストの巻末の「内蔵関数一覧表」を参照してください。

⑤PRINT文

文字列、数式、変数などの出力要素を画面に表示します。

⑥RETURN文

指定された行に、またはサブルーチンを呼び出した文の次の文に戻ります。

⑦GOTO文

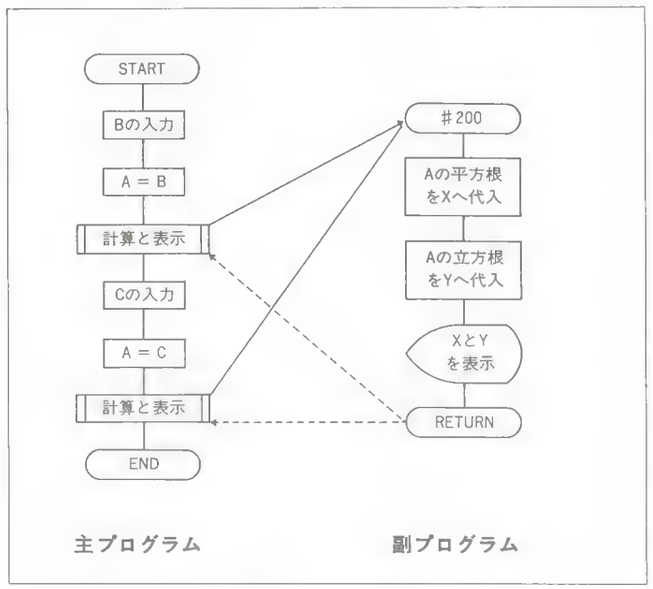
指定された行番号へ無条件でジャンプします。

⑧END

プログラムの実行を終了し、コマンドの入力待ちとなります。

コラム●サブルーチンの呼び出し●

ひとつのサブルーチンを、メインプログラムのいろいろな位置から何回も呼び出すことができます。



図のように呼び出す側のプログラムを主プログラム(メインルーチン)と呼び、呼び出される側のプログラムを副プログラム(サブルーチン)と呼びます。

また、サブルーチンのフローチャートは、メインルーチンのフローチャートとは別に書き、サブルーチンの最初の記号には、(サブルーチン)(＃200)(サブルーチン名)のようにサブルーチンの開始であることがわかるような名称を書き、最後の記号には、(RETURN)というように、メインプログラムへ戻ることがわかるような名称を書きます。

READ文とDATA文

ここで学ぶこと

☆READ文によるデータ入力と、DATA文によるデータの定義

まとめ

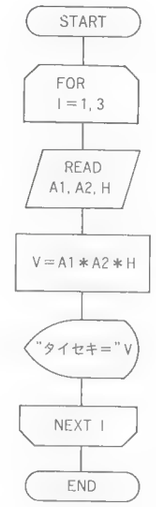
◎READ文は、DATA文で指定されたデータを順々に読み込みます。

◎DATA文は、READ文で読み込むデータを設定します。

プログラム例

```
① 10 FOR I=1 TO 3
② 20 READ A1,A2,H
④ 30 V=A1*A2*H
⑤ 40 PRINT "タイセキ=";V
① 50 NEXT I
⑥ 60 END
③ 70 DATA 3,5,6
③ 80 DATA 5,8,4
③ 90 DATA 3,4,10
```

(行番号70から90までのデータをREAD文で読み込み、変数A1、A2、Hに代入し、この変数によって直方体の体積を求めるプログラム)



プログラムフローチャート

プログラムの文法と要点

①FOR~NEXT文

制御変数を初期値から終値まで増分値で変化させながら、FOR文からNEXT文までの間に書かれた文を繰り返します。

②READ文

DATA文で定義されたデータを読み込み、指定した変数に順々に代入します。

```
        代入される変数名
          ↓  ↓  ↓
READ  A1、A2、H
```

③DATA文

READ文で読み込む数値または文字列を設定します。

```
        読み込まれるデータ
          ↓  ↓  ↓
DATA  3、5、6
```

これまで学んだプログラムでは、数値を扱う場合に、(1)LET文のように変数に代入する、(2)INPUT文を使用してキーボードから数値を入力する、という2つの方法がありました。

ここでは、もうひとつの方法として、READ文とDATA文を学びます。READ文を使用すれば、データをそのつど入力する必要がありません。プログラム中にすでに書き込まれていますので、後からデータのチェックを行なうときにも便利です。

```
READ  A1, A2, H
      ↑   ↑   ↑
DATA  3, 5, 6
DATA  5, 8, 4
DATA  3, 4, 10
```

READ文とDATA文は左図のように常に対になって使用され、READ文で読み込まれるデータ数あるいは読み込む回数と、DATA文中のデータ数とが対応していなければなりません。

もしREAD文で読み込まれるデータ数が、DATA文中に存在するデータ数よりも多い場合はエラーとなります。

```
READ  A, B, C, D, E
      ↑  ↑  ↑
DATA  10, 15, 20
```

A、B、Cにデータは読み込まれるが、D、Eに読み込まれるデータがない。

逆にREAD文で読み込まれるデータ数が、DATA文中に存在するデータ数よりも少ない場合はエラーとなりません。

```
READ  A, B, C, D, E
      ↑  ↑  ↑  ↑  ↑
DATA  5, 7, 3, 2, 1, 6, 4
```

6、4のデータは余る。

④代入文(LET文)

左辺の変数に右辺の値・文字または式の値を代入します。「LET」は省略してあります。

⑤PRINT文

文字列、数式、変数などの出力要素を画面に表示します。

⑥END

プログラムの実行を終了し、コマンドの入力待ちとなります。

コラム●読み込むデータの行番号指定●

READ文で読み込もうとするDATA文が複数組ある場合、同じREAD文を書かなくても、RESTORE文により読み込むデータの位置を指定することができます。

この場合、最初のREAD文が実行される前に「RESTORE 行番号」を指定すれば、指定された行番号のデータが読み込まれます。

配列変数の宣言

ここで学ぶこと

☆大量のデータを配列変数を使って処理をする

まとめ

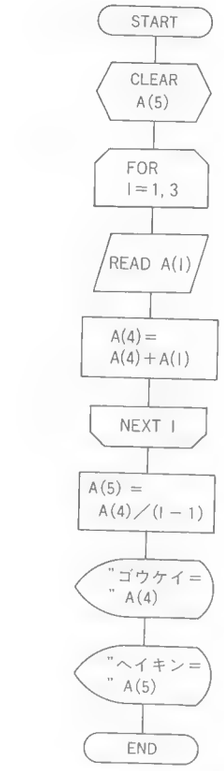
◎CLEAR文は、すべての変数をクリアーします。

◎DIM文は、配列変数を宣言します。

プログラム例

```
①    5  CLEAR
②   10  DIM A(5)
③   20  FOR I=1 TO 3
④   30  READ A(I)
⑥   40  A(4)=A(4)+A(I)
③   50  NEXT I
⑥   60  A(5)=A(4)/(I-1)
⑦   70  PRINT "ｺﾞｳｹｲ=";A(4)
⑦   80  PRINT "ﾍｲｷﾝ=";A(5)
⑧   90  END
⑤  100  DATA 3,2,10
```

(行番号100で指定した数値を配列に代入し、合計と平均を求めるプログラム)



プログラムフローチャート

プログラムの文法と要点

①CLEAR文

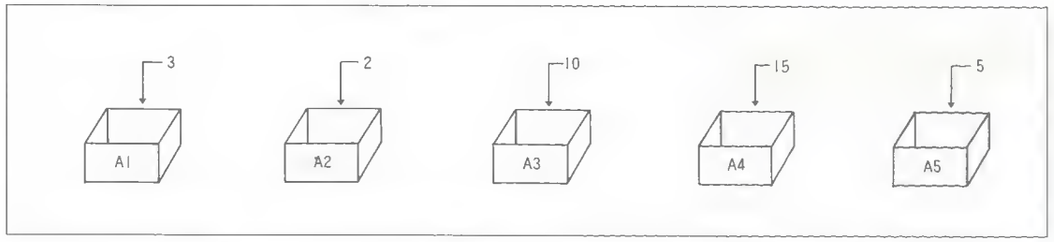
すべての変数をクリアーします。

②DIM文

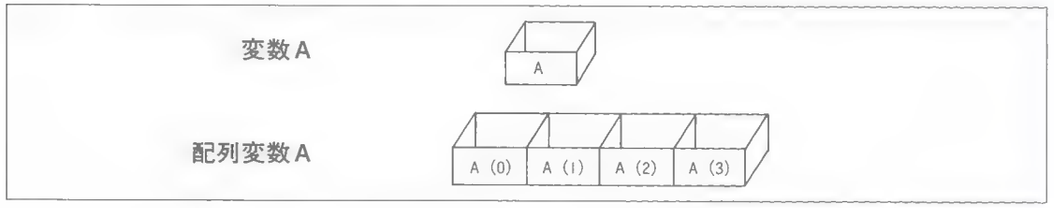
配列変数の宣言には「DIM」文を使います。

```
配列変数名  添え字の最大値
    ↓       ↓
DIM  A      (5)
```

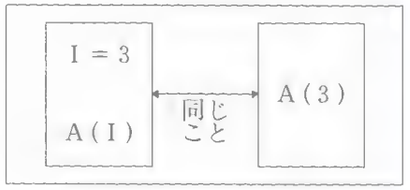
これまでに学んだ変数では、1つの変数に対して1つの値しか代入できません。ですから、数多くの値を扱うにはいくつもの変数を用意しなければなりませんでした。



配列を使うと１つの配列変数に対していくつもの値を代入することができます。
変数を箱に例えると、配列変数はいくつもの箱をひとまとめにしたものに例えられます。



１つ１つの箱を区別するために、配列変数名に続く（　）の中の数字で何番目の箱であるかを指定しています。この数字を「添え字」といいます。( )の中の添え字には、数値でも変数名でも指定することができます。(右の図を参照)



例題のプログラムでは、配列は５を宣言しています。配列の中にFOR～NEXT文で代入するよりも、READ文を１度実行して順に代入した方が良いと思う人がいるかも知れません。データ数が少ないときは、それでも大きな違いはありませんが、扱うデータが100個というような場合、１度のREAD文によるプログラムでは何倍もの長さになってしまいます。

③FOR～NEXT文

FOR文からNEXT文までの間に書かれた文を、制御変数を初期値から終値まで増分値で変化させながら繰り返します。

④READ文

DATA文で定義されたデータを読み込み、指定した変数に順々に代入します。

⑤DATA文

READ文で読み込む数値または文字列を設定します。

⑥代入文(LET文)

左辺の変数に右辺の値，文字または式の値を代入します。

行番号60の代入文では平均を求めています。なぜ変数Ｉより１を減算してから割るのかというと、FOR～NEXT文では制御変数が終値を超えたときに実行を終えるからです。

⑦PRINT文

文字列、数式、変数などの出力要素を表示します。

⑧END文

プログラムの実行を終了し、コマンドの入力待ちとなります。

コラム●配列の０番目？●

本機では配列を０番目から使用することができます。したがってA(5)という配列を宣言すると、０～５番目の計６個のデータを記憶することができます。

周辺機器との入出力

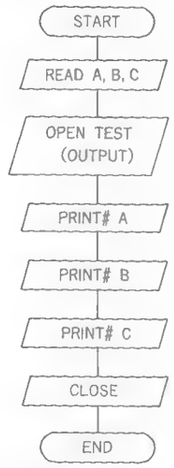
ここで学ぶこと
☆フロッピーディスクやカセットテープへの書き込み方法
☆フロッピーディスクやカセットテープからの読み込み方法

まとめ
◎OPEN文は、ファイルの使用を宣言します。
◎PRINT＃文は、指定されたファイルへデータを書き込みます。
◎CLOSE文は、ファイルの使用の終了を宣言します。
◎INPUT＃文は、指定されたファイルからデータを読み込みます。

プログラム例1

```
①  10  READ  A, B, C
②  20  DATA  17, 14, 21
③  30  OPEN "0:TEST" FOR OUTPUT
       AS #1
④  40  PRINT #1, A
④  50  PRINT #1, B
④  60  PRINT #1, C
⑤  70  CLOSE
⑥  80  END
```

(行番号20で指定した数値を変数に代入し、フロッピーディスクにオープンした"TEST"というファイルに変数の値を書き込むプログラム)

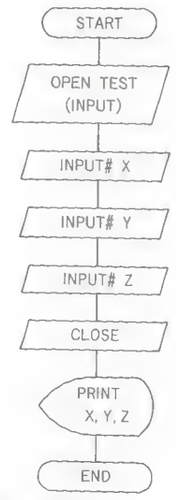


プログラムフローチャート

プログラム例2

```
③  10  OPEN "0:TEST" FOR INPUT
       AS #1
⑦  20  INPUT #1, X
⑦  30  INPUT #1, Y
⑦  40  INPUT #1, Z
⑤  50  CLOSE
⑧  60  PRINT X; Y; Z
⑥  70  END
```

(フロッピーディスクから"TEST"というファイルをオープンしてデータを読み込み表示するプログラム)



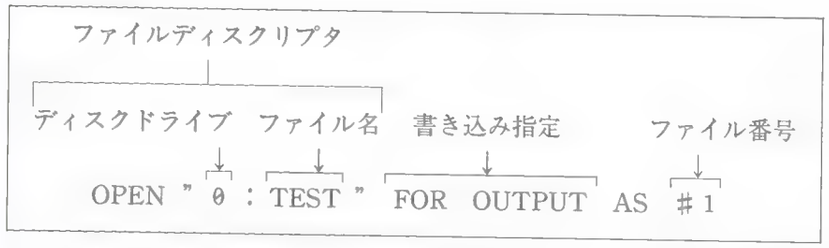
プログラムフローチャート

プログラムの文法と要点

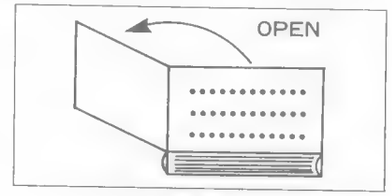
①READ文
DATA文で定義されたデータを読み込み、指定した変数に順々に代入します。

②DATA文
READ文で読み込む数値定数または文字定数を設定します。

③OPEN文
ファイルの使用を宣言します。プログラム例1、プログラム例2とも同じような宣言をしています。



フロッピーディスクやカセットテープに記録されたデータ、またはデータの集まりをファイルといいます。このファイルを使用する準備のための命令としてOPENがあります。OPEN文の実行は、これからデータを読み書きするデータファイルを「開く」という意味です。



ファイルのOPENのイメージ

ファイルを開く(OPEN)ということは、これからデータを記録するためノートや、データを読むための本などの表紙を「開く作業」と同じようなものと考えるとよいでしょう。

プログラム例1行番号30のOPEN文では、ディスクドライブの「TEST」という名前をつけたファイルに書き込みを行ないます。「FOR OUTPUT」という部分によって、OPENの目的が書き込みに指定されます。

OPENの目的は、次のように指定します。

	目　的	備　考
OPEN～FOR OUTPUT～	ファイルに書き込み	プログラム例1行番号30
INPUT	ファイルから読み込み	プログラム例2行番号10
APPEND	ファイルに追加書き込み	

OPEN文では、開いたファイルを間違えないようにするための目的として番号をつけています。プログラム例1、プログラム例2ではともにOPEN文では、ファイル番号として「AS　＃1」を指定しています。その後の出力するための命令（例1行番号40～60）と、入力するための命令（例2行番号20～40）で、同じくファイル番号として「＃1」を指定します。

④PRINT＃文

指定されたファイルに、出力要素を順番に出力します。

```
ファイル番号   出力要素(例では変数A)
          ↓ ↓
PRNIT   #1, A
```

PRINT＃文の実行によって、フロッピーディスクやカセットテープにデータが書き込まれます。そして、書き込まれたデータを読み込むには、INPUT＃文を使用します。プログラム例1で書き込まれたTESTというファイルを、プログラム例2で読み込んでいます。

フロッピーディスクやカセットテープに書かれたデータを読み込むときは、次のように指定してください。

注　意	書き込み	読み込み
数値型データは数値変数に読み込む	PRINT　#1, A	INPUT　#1, P
文字型データは文字変数に読み込む	PRINT　#1, B$	INPUT　#1, Q$
数値型データを文字変数に読み込んではいけない	PRINT　#1, C	INPUT　#1, R$
文字型データを数値変数に読み込んではいけない	PRINT　#1, D$	INPUT　#1, S
変数は繰り返し書き込み、繰り返し読み込める	PRINT　#1, E, F, G	INPUT　#1, T, U, V

コラム●BASICで扱うファイル媒体とファイルディスクリプタ●

本機で扱うことのできるファイルの媒体には、フロッピーディスク、カセットテープ、通信回線があります。これらを区別するために次のように指定します。

●フロッピーの場合　OPEN　"0：ファイル名" FOR～

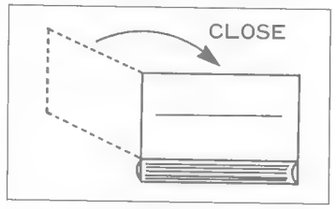
"　" で囲まれた部分をファイルディスクリプタといいます。ファイルディスクリプタは、次の種類があります。

対象となるファイル	ファイルディスクリプタ
フロッピーの指定	"CAS0：ファイル名"または"0：ファイル名"
カセットテープの指定	"CAS1：ファイル名"
通信回線の指定	"COM0：通信パラメータ"

⑤CLOSE文

ファイルを閉じ、入出力バッファ使用の終了を宣言します。CLOSE文では、ファイル名、出力モード、ファイル番号などは指定しません。

先ほどは書き込みそして読み込むための準備として、OPEN文を使ってファイルを開く作業をしました。書き込む、読み込むなどの作業が終了すれば、ファイルを閉じるという作業を行ないます。CLOSE文の実行は、データを読み書きしたデータファイルを「閉じる」という意味です。ファイルを閉じる（CLOSE）ということは、データを記録し終わったノートや、データを読み終わった本などの表紙を「閉じる作業」と同じと考えるとよいでしょう。



ファイルのCLOSEのイメージ

⑥END

プログラムの実行を終了し、コマンドの入力待ちとなります。

⑦INPUT＃文

指定されたファイルよりデータを読み込み、変数に代入します。

```
   ファイル番号 代入する変数名
          ↓ ↓
INPUT   #1, X
```

⑧PRINT文

文字列、数式、変数などの出力要素を表示します。

コラム●データの保存●

本機では電源を切っても内蔵の電池によってデータが保存されます。しかし、次の場合にはフロッピーディスクやカセットテープにデータを保存しておきましょう。

(1)本体の記憶容量が不足したとき

(2)本体に記憶されているデータと同じものを、バックアップとして保存しておきたいとき

このようなときは、PRINT＃文を使用してフロッピーディスクやカセットテープにデータを保存します。

コラム●データバンクを使う●

データバンクに書き込んだデータをBASICプログラム中で利用するために、READ＃文、RESTORE＃文、WRITE＃文が用意されています。

READ文はプログラム中のDATA文の内容を読み込みます。一方、READ＃文は、データバンク内のメモデータを指定する変数に読み込みます。

RESTORE文はREAD文で読み込むDATA文の位置を指定します。一方、RESTORE＃文は、メモデータをサーチ（検索）したり、READ＃文で読み込むメモデータの順番を変えたりします。

また、データバンクのメモデータの書き換えや削除をするための命令として、WRITE＃文が用意されています。

文法のまとめ

ここで学ぶこと
☆文、演算子、変数の復習

まとめ
◎文は、行番号とコマンドとオペランドで構成されます。
◎演算子には、比較演算子と文字演算子があります。
◎変数には、数値変数と文字変数と配列変数があります。

文の構成

各文(行)は、次のように構成されます。

行番号	コマンド(命令)	オペランド

比較演算子(プログラム中にのみ有効)

2つの数値もしくは2つの文字列の比較を行ない、結果を真(−1)または偽(0)で表わします。(演算子は「条件分岐するプログラム(IF～THEN～ELSE文)」を参照。)

	文	結　果	備　考
数値の比較	PRINT　123＞45	−1(真)	数値の大小を比較します。
	PRINT　123＜45	0(偽)	
文字列の比較	PRINT　"DEF"＜"ABC"	0(偽)	先頭から順に文字コードの大きさを比較します。
	PRINT　"ABC"＜"ABD"	−1(真)	
	PRINT　"ABC"＞"ABCD"	0(偽)	短い文字列(「ABC」の3文字)で比較し、等しいときは短い方が小と判断します。

文字演算子

文字列の演算は＋(プラス)だけが有効です。結果は、256文字以上の長さになってはいけません。

"A"＋"B"→"AB"
↑
255文字以下

変数の種類

- 変数
  - 数値変数
  - 文字変数
  - 配列変数
    - 配列数値変数
    - 配列文字変数

変数名と配列名の付け方

①英大文字、英小文字、カナで始まる文字列であること
②英大文字、英小文字、カナ、数字で構成されること
③先頭から予約語を含んではならないこと
④長さは255文字以内であること

配列

①配列は、DIM文で宣言します。
②配列の添え字は0以上の整数で、小数部は切り捨てられます。
③配列の次元は、内部スタックの許す範囲です。
④添え字の最大量は、記憶容量の許す範囲です。

変数、配列の取り扱い

①変数、配列は、すべてのプログラムに対して共通に用いられます。
②文字変数は、CLEAR文で指定される文字データ領域に格納されます。
③数値変数と文字変数は、初めて使用したときにその場所が確保されます。
④配列宣言をしないと、配列変数は使えません。

コラム●変数の使用バイト数●

数値変数と文字変数は、初めて使用したときにその場所が確保されます。それぞれの使用バイト数は次のようになります。

●数値変数：(変数名の長さ＋12)バイトがワークエリアから確保されます。
●文字変数：(変数名の長さ＋4)バイトがワークエリアから、また(文字列の長さ＋1)バイトが文字領域から確保されます。

配列変数は、DIM文で定義したときにその場所が確保されます。それぞれの使用バイト数は次のようになります。

●配列数値変数：{(変数名の長さ＋4)＋(配列の大きさ×8)＋次元×2＋1}バイトがワークエリアから確保されます。
●配列文字変数：(変数名の長さ＋4)バイトがワークエリアから、{(配列の大きさ)＋次元×2＋1}バイトが文字領域から確保されます。また、文字列が代入されたときに、その文字列の長さだけ文字領域を使用します。

マニュアルコマンド一覧表

注) ・マニュアルコマンドは、プログラム内で実行できません。
・{　}内はいずれか１つを示します。{　}そのものは書き込みません。
・〔　〕内は省略可能です。ただし〔　〕そのものは書き込みません。
・※の付いたコマンドは、CALモードでも使えます。

コマンド名	書　式	機能及び使用例
LIST	LIST {〔開始行番号〕〔－〔終了行番号〕〕 / 〔.〕 / 〔ALL〕}	プログラムの内容の全部または一部を画面に表示します。LISTをLLISTとすると、画面への出力からプリンタへの出力になります。 ⇨ ①LIST　(先頭から表示) ②LIST 30　(行番号30を表示) ③LIST 20-80　(行番号20～80を表示) ④LIST 20-　(行番号20以降を表示) ⑤LIST -90　(先頭行～行番号90までを表示) ⑥LIST.　(最後に扱った行を表示) ⑦LIST ALL　(全てのプログラムエリアのプログラムを表示)
RENUM	RENUM 〔新行番号〕〔,〔旧行番号〕〔,増分〕〕	行番号を一定間隔で付け直します。 ⇨RENUM 100,10,10(行番号10を行番号100とし以降10間隔で付け直します)
NEW	NEW 〔ALL〕	現在指定されているプログラムエリアのプログラムまたは、全てのプログラムエリアのプログラムを消去します。 ⇨ 78頁参照
※PASS	PASS〝パスワード〟	全プログラムエリアおよび全ファイルに対するパスワードを設定または解除します。⇨PASS〝CASIO〟
RUN	RUN〔開始行番号〕	プログラムを先頭行または指定した行から実行します。 ⇨ 81頁、85頁参照
SAVE	SAVE〔ALL〕〝ファイルディスクリプタ〟〔,A〕	現在指定されているプログラムエリアのプログラムまたは、全てのプログラムエリアのプログラムをファイルへ出力します。(ALL指定はカセットテープへの出力のときのみ行なえます。)「,A」を付けたときは、ASCII形式で出力します。 ⇨SAVE"DEMO1"(ファイル"DEMO1"へ出力)

コマンド名	書　式	機能及び使用例
LOAD	LOAD〔ALL〕〝ファイルディスクリプタ〟〔,A〕	現在指定されているプログラムエリアまたは、全てのプログラムエリアに、指定したファイルよりプログラムを読み込みます。(ALL指定はカセットテープから読み込むときのみ行なえます。)「,A」を付けたときは、ASCII形式のプログラムを読み込みます。 ⇨LOAD"DEMO1"(ファイル"DEMO1"より読み込みます)
MERGE	MERGE"ファイルディスクリプタ"	現在指定されているプログラムエリアに、ファイルディスクリプタで指定されたファイル上のプログラムを混合します。 ⇨①MERGE"CAS0:TEST"(カセットテープ内ファイル"TEST"と混合) ②MERGE"0:TEST.BAS"(フロッピーディスク内ファイル"TEST.BAS"と混合)
VERIFY	VERIFY"ファイルディスクリプタ"	カセットテープに記録されているファイルをチェックします。 ⇨VERIFY"CAC0:TEST"(ファイル"TEST"が正しく記憶されているか)
EDIT	EDIT {〔行番号〕 / 〔.〕}	現在指定されているプログラムエリアのプログラムを表示し、編集モードに入ります。 ⇨①EDIT　(最初の行より編集を行ないます) ②EDIT 30　(行番号30の編集を行ないます) ③EDIT .　(最後に扱った行の編集を行ないます)
DELETE	DELETE 〔開始行番号〕〔－〔終了行番号〕〕	プログラムの一部を行単位で削除します。(パラメータを両方とも省略することはできません) ⇨①DELETE 50　(行番号50を削除) ②DELETE 20-80　(行番号20～80を削除) ③DELETE 20-　(行番号20以降を削除) ④DELETE -90　(先頭行～行番号90までを削除)

コマンド名	書　式	機能及び使用例
*SYSTEM	SYSTEM	プリンタのon／off設定、トレースモードのon／off設定、CLEAR文の設定、各領域のフリーエリア容量を表示します。 ⇨SYSTEM(プリンタ:PR、トレース:TR、テキストエリアのフリー：FREE、数値変数領域のフリー：V、文字変数領域のフリー：$が表示されます。)
*CONT	CONT	STOP文またはSTOPキーで停止されたプログラムの実行を再開します。 ⇨CONT
*LIST＃	LIST＃	データバンクエリア"F0"に書き込まれているメモデータをすべて表示します。LISTをLLISTとすると、画面への出力からプリンタへの出力になります。 ⇨LIST＃
*SAVE＃	SAVE＃〔"ファイルディスクリプタ"〕	データバンクエリア"F0"に書き込まれているメモデータをすべてファイルディスクリプタで指定されているファイルへ出力します。 ⇨SAVE＃"CAS0:TEST"(カセットテープ内ファイル"TEST"に出力)
*LOAD＃	LOAD＃〔"ファイルディスクリプタ"〕	データバンクエリア"F0"に、ファイルディスクリプタで指定されているファイルよりメモデータをすべて読み込みます。 ⇨LOAD＃"CAS0：TEST"(カセットテープ内ファイル"TEST"より読み込む)
*MERGE＃	MERGE＃〔"ファイルディスクリプタ"〕	データバンクエリア"F0"に、ファイルディスクリプタで指定されているファイルよりメモデータを追加します。 ⇨①MERGE＃"CAS0:TEST"(カセットテープ内ファイル"TEST"より追加) ②MERGE＃"0:TEST.BAS"(フロッピーディスク内ファイル"TEST.BAS"より追加)
*NEW＃	NEW＃	データバンクエリア"F0"に書き込まれているメモデータを全て消去します。 ⇨NEW＃

プログラムコマンド一覧表

注）・｛　｝内はいずれか１つを示します。｛　｝そのものは書き込みません。
　　・〔　〕内は省略可能です。ただし〔　〕そのものは書き込みません。

コマンド名	書　式	機能及び使用例
BEEP	BEEP ｛〔0〕 1｝	ブザーを鳴らします。 ⇨①BEEP　(低音で鳴らします) ②BEEP0 (低音で鳴らします) ③BEEP1 (高音で鳴らします)(86頁参照)
CLEAR	CLEAR〔文字領域サイズ〕 〔，ワーク領域サイズ〕	すべての変数をクリアーし、パラメータに従ってメモリーの領域を確保します。 ⇨CLEAR (97頁参照)
DIM	DIM 配列名 (添え字の最大値 〔，添え字の最大値…〕)	配列の変数を宣言します。 ⇨①DIM A(5)　(数値変数Aを１次元の配列宣言) ②DIM B$(2,5) (文字変数B$を2次元の配列宣言) (97頁参照)
ERASE	ERASE 配列名〔，配列名)	指定した配列を変数名単位で消去します。 ⇨ERASE A、B
END	END	プログラムを終了します。 ⇨END
DATA	DATA　データ1〔，データ2…〕	READ文で読むデータを設定します。 ⇨DATA 10,20,15 (95頁参照)
READ	READ 変数1〔,変数2〕	DATA文で設定されたデータを読み込み、変数に順次格納します。 ⇨READ A, B, C　(95頁参照)
RESTORE	RESTORE〔行番号〕	READ文で読むデータ文の開始行を指定します。 ⇨RESTORE 100
FOR～TO ～STEP ⋮ NEXT	FOR　変数＝初期値　TO　終値 〔STEP　増分値〕 ⋮ NEXT〔変数名〕	FOR～NEXT間を変数が終値を超えない範囲で繰り返し行ないます。 ⇨FOR　I＝1　TO 10 ⋮ NEXT　I　(89頁参照)
GOTO	GOTO ｛分岐先行番号 ＃プログラムエリア番号	指定された分岐先行番号へ、または指定されたプログラムエリアの先頭の行番号へ、無条件でジャンプします。 ⇨①GOTO 80 ②GOTO ＃7　(91頁参照)

コマンド名	書　式	機能及び使用例
GOSUB	GOSUB {分岐先行番号 / #プログラムエリア番号}	指定された分岐先行番号からはじまるサブルーチンを、または指定されたプログラムエリアの先頭からはじまるサブルーチンを呼び出します。 ⇨①GOSUB 100 ②GOSUB #5　(93頁参照)
RETURN	RETURN [{戻り先行番号 / #プログラムエリア番号}]	サブルーチンから指定された行番号または、指定されたプログラムエリアの先頭に復帰します。戻り先行番号の指定がないときは、サブルーチンを呼んだ文の次の文へ復帰します。 ⇨①RETURN ②RETURN 20 ③RETURN #1　(93頁参照)
IF～ (THEN / GOTO)～ ELSE	IF条件文 { THEN {文[:文] / 分岐先行番号 / #プログラムエリア番号} GOTO {分岐先行番号 / #プログラムエリア番号} [ELSE {文[:文] / 分岐先行番号 / #プログラムエリア番号}]	条件式が成立した時、THEN以下の文を実行するか、THEN、GOTO以下の分岐先行番号へまたは指定されたプログラムエリアの先頭の行番号へジャンプします。また条件式が成立しないときはELSE以下の文を実行するか、ELSE以下の分岐先行番号へ、または指定されたプログラムエリアの先頭の行番号へジャンプします。 ⇨①IF A>100 THEN 50 ELSE 80 ②IF B=0 THEN X=B ELSE Y=B ③IF C<>50 THEN #9
INPUT	INPUT["メッセージ文"{;/,}] 変数名 [,["メッセージ文"{;/,}]変数名…]	キーボードからデータを入力し、指定した変数に代入します。また、メッセージ文を表示し変数を入力することもできます。 ⇨①INPUT A, B, C ②INPUT "X=";X　(86頁参照)
LET	LET 変数名＝{代入値 / 計算値}	左辺の変数名に右辺の値または、計算式の値を代入します。代入文は、LET自体を省略することができます。 ⇨①LET A=10 ③A$="カシオ" ④X=Y*Z/2　(86頁参照)

コメント名	書　式	機能及び使用例
ON～GOTO	ON 数式 GOTO {分岐先行番号 / #プログラムエリア番号} [,{分岐先行番号 / #プログラムエリア番号}…]	ON以下の数式の値に対応して、GOTO文を実行します。分岐先は先頭から順に式の値が1の場合、2の場合…と割り当てられます。 ⇨①ON A GOTO 100,200,300 ②ON A+B GOTO 100,#6,#7
ON～ GOSUB	ON 数式 GOSUB {分岐先行番号 / #プログラムエリア番号} [,{分岐先行番号 / #プログラムエリア番号}…]	ON以下の数式の値に対応して、指定された分岐先行番号からはじまるサブルーチンを、または指定されたプログラムエリアの先頭からはじまるサブルーチンを呼び出します。分岐先は先頭から順に式の値が1の場合、2の場合…と割り当てられます。 ⇨①ON A GOSUB 1000,1200,1500 ②ON A+B GOSUB 100,#5,#8
PRINT	PRINT [{TAB(タブ指定) / 数式 / 文字列 / 変数}] [{;/,} {TAB(タブ指定) / 数式 / 文字列 / 変数}…]	変数の内容、式の値、文字列などの出力要素を表示します。PRINTをLPRINTとすると、画面の出力からプリンタへの出力になります。 ⇨①PRINT　(改行) ②PRINT A, B, C (変数A, B, Cの内容表示) ③PRINT "X=";X(文字列と変数Xの内容表示) ④PRINT TAB(5);"カシオ"(文字列を左端より5つずつずらして表示) (86頁参照)
REM	{REM / '} 注釈文	注釈(コメント)を表わし、何も実行しません。 ⇨REM サイダイチ ヲ モトメル プログラム
SET	SET {F / E / N} 0～9の1文字	数値データの出力形式を指定します。Fは小数点以下の桁数を、Eは有効桁数を指定し、Nは指定を解除します。 ⇨SET F3
STOP	STOP	プログラムの実行を一時停止します。CONT文で実行を再開します。 ⇨STOP　(85頁参照)

コマンド名	書　式	機能及び使用例
READ＃	READ＃ 変数名 〔, 変数名…〕	データバンクエリア"F0"に書き込まれているメモデータを変数に読み込みます。 ⇨READ＃ A$, X
WRITE＃	WRITE＃ 〔メモデータ〕 〔, メモデータ…〕	データバンクエリア"F0"のデータの書き換え削除を行ないます。 ⇨①WRITE＃ (削除) ②WRITE＃"カシオ VX-4"(書き換え)
RESTORE＃	RESTORE＃ 〔("ファイルエリア名")〕 〔"検索文字列"〕 〔, {0/1}〕〔,GOTO{分岐先行番号 / #プログラムエリア番号}〕	READ＃、WRITE＃、の対象になるファイルエリアを切り換えます。また、ファイルエリア内のデータを検索し、READ＃文で読むデータの位置を変更します。 ⇨①RESTORE＃("F1") ②RESTORE＃"START"
CLOSE	CLOSE	現在使用しているファイルを閉じて、I/Oバッファ使用の終了を宣言します。 ⇨CLOSE
CLS	CLS	ディスプレイ画面を消去(クリアー)します。 ⇨CLS
DEFSEG	DEFSEG＝セグメントアドレス	PEEK関数、POKE文を行なうときのセグメントベースアドレスを設定します。 ⇨DEFSEG＝16
LOCATE	LOCATE X座標, Y座標	仮想スクリーン上のカーソルの位置を指定します。 ⇨LOCATE 10,0
DEFCHR$	DEFCHR$(コード)＝"文字形"	表示する文字の字形を定義します。 ⇨DEFCHR$(252)＝"0F0F0F0F0F0F"
POKE	ROKE アドレス, データ	指定されたアドレスにデータを書き込みます。 ⇨POKE & H7000,0
TRON	TRON	BACICプログラムのトレースモードを設定します。 ⇨TRON (85頁参照)
TROFF	TROFF	BACICプログラムのトレースモードを解除します。 ⇨TROFF (85頁参照)
VARLIST	VARLIST	現時点で存在するすべての変数名、配列名を表示します。 ⇨VARLIST

コマンド名	書　式	機能及び使用例
INPUT＃	INPUT＃ ファイル番号, 変数名 〔, 変数名…〕	指定されたファイル番号のシーケンシャルファイルからデータを読み込みます。 ⇨INPUT #1, A (100頁参照)
ON ERROR GOTO	ON ERROR GOTO 分岐先行番号	エラー発生時の分岐先行番号を指定します。
OPEN	OPEN"ファイルディスクリプタ" 〔FOR{INPUT / OUTPUT / APPEND}〕AS〔＃〕 ファイル番号	ファイルの使用を宣言します。 ⇨OPEN "DATA1" FOR OUTPUT AS ＃1
PRINT＃	PRINT＃ ファイル番号〔,{TAB(タブ指定) / 数式 / 文字式}〕 〔{; / ,}{TAB(タブ指定) / 数式 / 文字式}…〕〔{; / ,}〕	シーケンシャルファイルへ、変数の内容、式の値文字列などの出力要素を出力します。 ⇨PRINT＃1, A$ (100頁参照)
RESUME	RESUME〔{NEXT / 戻り行番号}〕	エラー処理ルーチンより復帰します。(NEXTまたは戻り番号を省略すると、エラーの発生した文へ戻ります) ⇨①RESUME NEXT (エラーの文の次の文へ復帰) ②RESUME 100
FORMAT	FORMAT	フロッピーディスクをフォーマット (初期化)します。 ⇨FORMAT
FILES	FILES"ファイルディスクリプタ"	フロッピーディスク内のファイルディスクリプタで指定されたファイル名、属性、使用容量などを表示します。 ⇨①FILES ②FILES "0:TEST. DAT" ③FILES "0:＊.DAT"
KILL	KILL "ファイルディスクリプタ"	フロッピーディスク内のファイルディスクリプタで指定されたファイルを消去します。 ⇨①KILL "0:TEST. DAT" ②KILL "0:＊.DAT"
NAME	NAME "旧ファイルディスクリプタ" AS "新ファイルディスクリプタ"	フロッピーディスク内のファイル名を変更します。 ⇨NAME "0:TEST. BAS" AS "0:ABC. BAS"

コマンド名	書　式	機能及び使用例
CHAIN	CHAIN"ファイルディスクリプタ"	現在のプログラムエリアにファイルディスクリプタで指定されたファイル上のプログラムの読み込みを実行します。 ⇨①CHAIN " CAS0:TEST" ②CHAIN "0:TEST.BAS"
LINE INPUT#	LINE INPUT# ファイル番号，文字変数名	シーケンシャルファイルから、1行文のデータを読み込みます。 ⇨LINE INPUT #1,A$
STAT	STAT Xデータ 〔,Yデータ〕〔;度数〕	統計データを入力します。 ⇨STAT 1,3;10
STAT CLEAR	STAT CLEAR	統計処理機能をクリアー(初期化)します。 ⇨STAT CLEAR
PRINT USING	PRINT USING "書式指定";出力要素	書式指定に従って、出力要素を表示します。なおUSING以下はLPRINT、PRINT#にも指定することができます。 ⇨①PRINT USING"& &";A$ (& &の長さ分のみ表示されます) ②PRINT USING"###.##";X (###.##にあわせて数値を表示し、整数部分の無効数字は空白に置き換えられます)

第3章 CASL編

簡単なプログラム

ここで学ぶこと
☆ソースプログラム
☆ソースプログラムの入力

まとめ

◎CASLで書かれたプログラムを「ソースプログラム」といいます。
◎ソースプログラムを作成・編集するにはCASLのメニュー画面でSキーを押します。

ソースプログラム

アセンブラ言語で書かれたプログラムのことを「ソースプログラム」といいます。このソースプログラムを機械語に翻訳するものが「アセンブラ」です。

ソースプログラムの入力

[プログラム例] 挨拶のプログラム

ラベル	命令コード	オペランド	説　明
REI	START	STAT	プログラムをSTATから開始
STAT	OUT	CHR, N5	CHR番地から(N5)=5文字分の出力
	EXIT		プログラムの実行を終了
N5	DC	5	出力する文字数5を格納
CHR	DC	'コンニチハ'	出力する文字列を格納
	END		プログラムの終了

このプログラムは、CHR番地から格納されている文字列「コンニチハ」を画面に出力するプログラムです。このプログラムを入力して実行してみましょう。

[操作方法]

電源をONにして、CASLモードに入ります。

操作　ON CASL

```
〈 CASL 〉

F 0 1 2 3 4 5 6 7 8 9          3355B
F1>Assemble/Source/Cal
```

CASLのメニュー画面になりました。このメニューから次のような機能が選択できます。

Aキー(Assemble)…ソースプログラムのアセンブル
Sキー(Source)……ソースの作成・編集
Cキー(Cal)……マニュアル計算モードへ戻る

まずソースプログラムを入力するため、Sキーを押します。

操作　S

```
■
                                〈  1〉
```

CASLエディタに入り、入力待ちの状態になりました。

操作　(1行目)　R E I TAB S T A R T TAB S T A T EXE
(2行目)　S T A T TAB O U T TAB C H R , N 5 EXE
(3行目)　TAB E X I T EXE
(4行目)　N 5 TAB D C TAB 5 EXE
(5行目)　C H R TAB D C TAB SHIFT ' SHIFT カナ K O N N I T I H A SHIFT カナ SHIFT ' EXE
(6行目)　TAB E N D EXE

これでプログラムが入力できました。

コラム●TABキーでプログラムを読みやすく●

CASLでは、「ラベル」「命令」「オペランド」の間に1文字以上の空白を空ける必要があります。また、「ラベル」を省略する場合は行頭に1文字以上の空白を空けなければなりません。

このとき、TABキーを利用して空白をつくると便利です。これにより、「ラベル」、「命令」、「オペランド」の先頭がそろった見やすいプログラムになります。TABキーはENGキーの下に水色の文字で書かれています。CASLモードでは、このキーはTABキーとしてだけ機能します。

ソースプログラムの編集・訂正・アセンブル

ここで学ぶこと
☆ソースプログラムの訂正
☆プログラムのアセンブル

まとめ
◎メニューに戻りたいときはCASLキーを押します。
◎ソースプログラムを訂正したいときは⇧⇩キーで行を選択し、⇦⇨キーで訂正箇所に移動します。
◎ソースプログラムをアセンブルすることにより、実行可能なオブジェクトプログラムができあがります。

CASLキー
ソースの編集中にメニューに戻るときにはCASLキーを押します。

ソースの訂正

ソースプログラムの訂正には以下のような方法があります。

訂正	⇧⇩⇦⇨を使って、訂正箇所にカーソルを移動させます。
1行削除	⇧⇩を使って削除したい行にカーソルを移動させ、SHIFT LOAD/SAVEと押します。
行の挿入	⇧⇩⇦⇨を使って挿入したい所にカーソルを移動させ、挿入したい文字を入力した後にEXEを押します。
行の検索	SHIFT LINE行番号EXEと押します。
文字列の検索	MEMO検索文字列EXEと押します。

ソースプログラムのアセンブル

CASLで書かれたソースプログラムをコンピュータが実行できるオブジェクトプログラムに翻訳することを、「アセンブル」といいます。

アセンブルするには、メニュー画面でAキーを押します。

操作 A

```
〈 CASL 〉

Go/Dump/Source/Cal/Print/Trace
PC:0000  REI          [off] [off]
```

アセンブルが正常に終了すると、上の画面が表示されます。エラーが発生した場合は 121ページを参照してください。

オブジェクトプログラム

ここで学ぶこと
☆オブジェクトプログラム
☆アセンブル後のメニュー
☆オブジェクトプログラムの実行

まとめ
◎ソースプログラムをアセンブルしたものを「オブジェクトプログラム」といいます。
◎アセンブルが正常終了すると、画面に実行のために必要なメニューが現われます。
◎オブジェクトプロクラムを実行するためには2つの方法があります。

オブジェクトプログラム

ソースプログラムをアセンブラによって機械語に翻訳したものを、オブジェクトプログラムといいます。オブジェクトプログラムは、コンピュータが直接理解できるプログラムです。

アセンブル後のメニュー

ソースプログラムをアセンブルして、エラーを生じることなしに正常に終了した場合は、画面は右のように変わります。

```
〈 CASL 〉

Go/Dump/Source/Cal/Print/Trace
PC:0000  REI          [off] [off]
```

それぞれのキーには、次のような機能が割り当てられています。

Gキー(Go)	オブジェクトプログラムの実行画面に移ります。
Dキー(Dump)	ダンプのメニュー画面に移ります。
Sキー(Source)	エディタに入り、ソースプログラムを表示します。
Cキー(Cal)	マニュアル計算モードに入ります。
Pキー(Print)	プリンタのON／OFFを指定します。
Tキー(Trace)	トレースのON／OFFを指定します。
EXEキー(実行)	オブジェクトプログラムを実行します。

オブジェクトプログラムの実行

オブジェクトプログラムを実行するには、次の２つの方法があります。ここでは116ページのプログラムを実行しましょう。

(a)　[EXE]キーによる実行

[EXE]キーを押すと、右のように画面が変わります。この画面で[EXE]キーを押すと、メニューの画面に戻ります。

```
ｺﾝﾆﾁﾊ
```

(b)　[G]キーによる実行

[G]キーを押すと右のように画面が変わります。ここで実行開始アドレスを入力し、[EXE]キーを押すと実行を開始します。

```
PC:0000  REI

go?_
```

実行開始アドレスを省略した場合は、最上行に表示されているアドレスから実行されます。また、アドレス入力は10進、16進、ラベル入力の３種類が可能です。

エラーメッセージ

ここで学ぶこと
☆アセンブル時のエラー
☆オブジェクトプログラム実行時のエラー

まとめ
◎ソースプログラムに誤りがあると、アセンブルしたときにさまざまな種類のエラーが発生します。
◎オブジェクトプログラムを実行中にも、メモリーの容量の不足などでエラーが生じます。

アセンブル時のエラー

ソースプログラムに誤りがあって、アセンブルできないときはエラーを検出して次のようにエラーメッセージを表示します。

エラーメッセージ	エ　ラ　ー　内　容
OMerror	オブジェクトを作成するためのメモリー容量が確保できない
LA error	ラベルの二重定義などラベル指定の誤り
OC error	オペコードエラー（命令語の誤り）
OR error	オペランドエラー
SO error	STARTやENDがない、または書式が違うなどのソースエラー

オブジェクトプログラム実行時のエラー

実行中にエラーが生じたときには、エラーを検出して次のようなエラーメッセージを出力します。

エラーメッセージ	エ　ラ　ー　内　容
Bad code	オブジェクト中に実行できないコードが入ってきた
Out of address	確保していないアドレスにアクセスしようとした
Stack over	スタックエリアがオーバーした

エラーメッセージが表示されたときは、[EXE]キーを押してメニュー画面に戻り、[S]キーを押してCASLエディタに入ってソースプログラムを修正してください。

割り算のプログラム

ここで学ぶこと
☆割り算のプログラム

まとめ
◎ここでは実用的なプログラムの例として、割り算のプログラムをあげます。

［プログラム例］ 割り算のプログラム（A÷B＝SHO余りAMARI）

ラベル	命令コード	オペランド	説　明
JOZAN	START		割り算(除算)プログラムを開始
	LEA	GR1, 0	GR1に商を入れる、初期値0
	LD	GR0, A	GR0にA番地の内容を入れる
LOOP	SUB	GR0, B	GR0からB番地の内容を引く
	JMI	ANS	結果が負ならばANS番地に飛ぶ
	LEA	GR1, 1, GR1	GR1に１を加える
	JMP	LOOP	LOOP番地に飛ぶ
ANS	ADD	GR0, B	GR0にB番地の内容を加える
	ST	GR0, AMARI	GR0を余りとしてAMARIに格納
	ST	GR1, SHO	GR1を商としてSHOに格納
	EXIT		プログラムの実行終了
A	DC	13	割られる数13
B	DC	5	割る数5
SHO	DS	1	商を格納する番地
AMARI	DS	1	余りを格納する番地
	END		プログラムの終了

割り算のアルゴリズム

CASLには割り算を直接行なう命令コードがないために、割り算を行なうためには引き算を繰り返し行ないます。

このプログラムは、A÷Bを行ない商と余りを出すプログラムです。

上の表の例では、割られる数13がA番地に格納されており、割る数5がB番地に格納されています。したがって13÷5を行ない、結果として商が2、余りが3になるわけです。(13÷5＝2余り3)

このプログラムでは、割られる数をGR0に入れ、LOOP行からの4行で割られる数13から割る数5を順次引いていき、引けなくなったら（引いた結果が負になってしまったら）ANS番地にジャンプし、最後に結果をSHO番地とAMARI番地に格納します。

プログラムを入力して、アセンブルし実行してみてください。エラーが発生した場合は、CASLエディタでソースプログラムを修正します。

ご注意
◎このプログラムを次ページ以降の「プログラムのダンプ」、「プログラムのトレース」のプログラム例にします。

プログラムのダンプ

ここで学ぶこと
☆ダンプ
☆オブジェクトプログラムのダンプ
☆ブレークポイントの設定

まとめ
◎主記憶装置の内容やレジスタの内容を出力することを、ダンプ(Dump)といいます。
◎ダンプのメニュー画面でOキーを押すと、オブジェクトプログラムのダンプができます。
◎ダンプのメニュー画面でRキーを押すとレジスタのダンプができます。
◎ダンプのメニュー画面でBキーを押すと、ブレイクポイントの設定ができます。

ダンプ

主記憶装置の内容やレジスタの内容を出力することをダンプ(Dump)といいます。これにより、オブジェクトプログラムを直接目でみたり、レジスタの内容を確認することができます。ダンプの指定は、アセンブルした後のメニュー画面で行ないます。

この画面でDキー(Dump)を押すと、ダンプのメニューが表示されます。

アセンブル後のメニュー

```
〈 CASL 〉
Go/Dump/Source/Cal/Print/Trace
PC:0000  JOZAN     [off] [off]
```

ダンプメニュー

```
〈 CASL 〉
ObJect/Register/Break point
```

オブジェクトプログラムのダンプ

ダンプのメニュー画面でOキーを押すと、オブジェクトプログラムのダンプができます。ここでアドレス値を入力してEXEキーを押すと、そのアドレス値からオブジェクトの内容を表示します。アドレス入力を省略してEXEキーを押すと0番地から表示します。

ダンプのキー操作

EXEまたは↓キー	下スクロール
↑キー	上スクロール

オブジェクトのダンプ

```
0000      JOZAN    1210
0001               0000
0002               1000
0003               0014
```

レジスタのダンプ

ダンプのメニュー画面で[R]キーを押すと、レジスタのダンプができます。

下の画面はレジスタの初期値を示しています。ここで割り算プログラムを実行して、レジスタの変化を見てみましょう。

レジスタの初期値

```
GR0:0000  0          GR1:0000  0
GR2:0000  0          GR3:0000  0
GR4:FFFF  65535      FR  :0
GR0?_
```

まず、[CASL]キーでCASLのメニュー画面に戻り、アセンブルし[G][EXE]キーでプログラムを実行します。そして[D]キーのあとに[R]キーを押すとダンプの変化を見ることができます。

レジスタの変化値

```
GR0:0003  3          GR1:0002  2
GR2:0000  0          GR3:0000  0
GR4:FFFF  65535      FR  :0
GR0?_
```

ブレークポイントの設定

ダンプのメニュー画面で[B]キーを押すと、ブレークポイントの設定ができます。ブレークポイントを設定することによって、オブジェクトプログラムの実行を中断させ、その時点でのレジスタとフラグの値を直接見ることができます。

ブレークポイントの解除は、アドレス入力時に[−][1][EXE]と入力します。

(ご注意)

◎このページの説明は、前ページの「割り算のプログラム」をアセンブルし、ダンプしたものです。

コラム●オブジェクトとレジスタのダンプの訂正●

オブジェクトとレジスタのダンプを訂正して、再実行させることができます。この方法では、ソースプログラムの内容に関わらず実行されます。ダンプを訂正したときは、ソースプログラムの予想される実行結果と異なる結果が現われますのでご注意ください。

レジスタダンプの訂正

レジスタダンプを行なうと、画面の最下行に「GR0?_」というメッセージが現われます。ここで16進数を入力し[EXE]キーを押すと、GR0にデータを与えることができます。訂正しないときは、[EXE]キーだけを押します。このように、次々とGR1～GR4までにデータを与えることができます。

オブジェクトの訂正

オブジェクトのダンプを表示しているときに[⇨]キーまたは[⇦]キーを押すと、画面の最下行に「data?_」というメッセージが現われます。ここで、16進数を入力し[EXE]キーを押すと、画面の最上行に表示されているアドレスのオブジェクト訂正ができます。訂正が終わったら[BRK]キーを押してメニュー画面に戻って実行すると、訂正後のオブジェクトが実行されます。

プログラムのトレース

ここで学ぶこと
☆トレースの方法

(まとめ)
◎オブジェクトプログラムの実行過程のレジスタ値とフラグの値を、1ステップずつ見ることが「トレース」です。

トレース

オブジェクトプログラムの実行過程のレジスタ値とフラグの値を、1ステップずつ見ることを「トレース」といいます。本機のCASLでは、表示画面またはプリンタにトレースの結果を出力することができます。

トレースの指定は、CASLのメニュー画面で行ないます。

(操作) [T]キーを押します。

```
〈 CASL 〉
Go/Dump/Source/Cal/Print/Trace
PC:0000   JOZAN      [off] [on]
```

トレースの指定がされると上のような画面になります。この状態でオブジェクトプログラムを実行すると、1ステップずつ、実行過程を表示画面で見ることができます。

なお、このとき[P]キーを押してPrintを〔on〕としておけば、トレースの内容はプリンタにも出力されます。

ここでは、「割り算プログラム」を実行したトレース結果の一部を次に示しておきます。

(操作) [G][EXE]

```
0000  JOZAN  LEA   GR1,0000
      0000 0000 0000 0000 FFFF 1
[EXE] 0002        LD    GR0,0014
      000D 0000 0000 0000 FFFF 1
[EXE] 0004  LOOP  SUB   GR0,0015
      0008 0000 0000 0000 FFFF 0
[EXE] 0006        JMI   000C
      0008 0000 0000 0000 FFFF 0
[EXE] 0008        LEA   GR1,0001,GR1
      0008 0001 0000 0000 FFFF 0
[EXE] 000A        JMP   0004
      0008 0001 0000 0000 FFFF 0
[EXE] 0004  LOOP  SUB   GR0,0015
      0003 0001 0000 0000 FFFF 0
                ⋮
      以下、表示は省略します。
```

表示の見方

```
PC   ラベル   命令コード オペランド
↓      ↓       ↓        ↓
0000  JOZAN   LEA   GR1,0000
      0000  0000  0000  0000  FFFF  1
        ↑     ↑     ↑     ↑     ↑    ↑
      GR0   GR1   GR2   GR3   GR4   FR
```

GR0
〜 } レジスタの内容
GR4

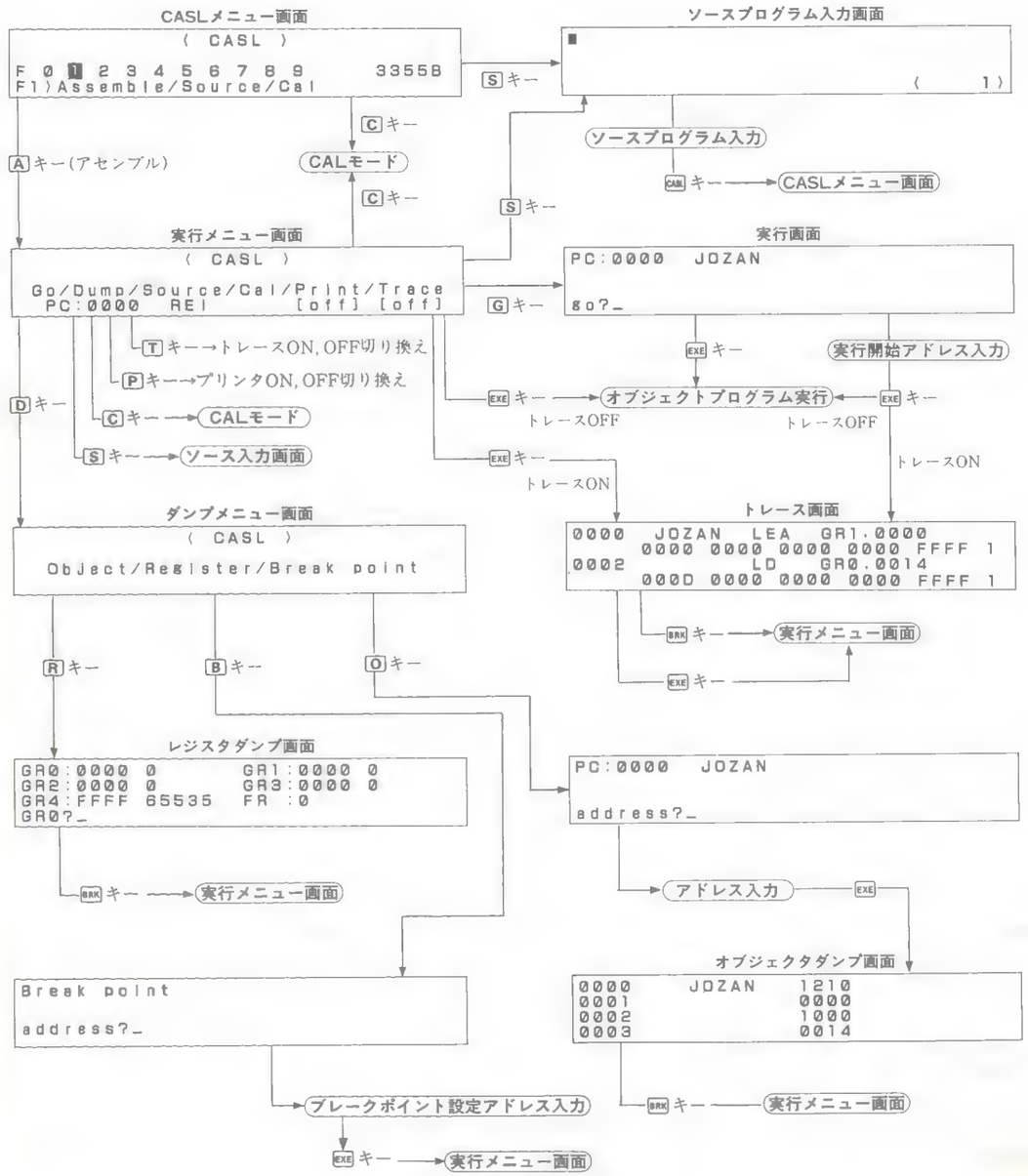
表　示	本機	仕様書
FR　フラグの値	0	00
	1	01
	2	10

仕様書とは試験センターのCOMET仕様書のことです。

操作のまとめ

ここで学ぶこと
☆CASLの基本操作

まとめ
◎CASLモードの手順は、まずソースプログラムを作り、デバッグして、アセンブルしたあとで、オブジェクトプログラムを実行させます。



CASLの紹介

ここで学ぶこと
☆コンピュータシステム
☆COMET
☆CASL

まとめ
◎コンピュータシステムには、ハードウェアとソフトウェアがあります。
◎COMETは通産省の情報処理技術者試験用に考えられた仮想コンピュータです。
◎CASLはCOMETで動くアセンブラ言語です。

コンピュータシステム

コンピュータシステムには、「ハードウェア」と「ソフトウェア」があります。

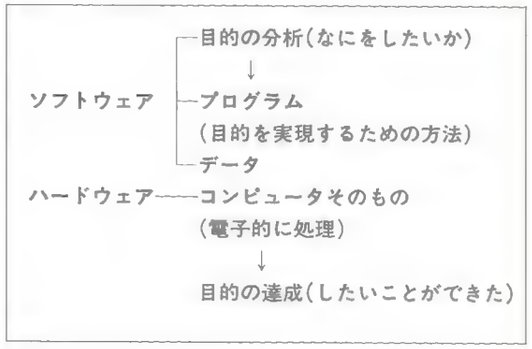
ハードウェアは、「コンピュータそのもの」「コンピュータの構成要素」「コンピュータを形作るLSI（大規模集積回路）やその他の物理的構成要素」といった幅広い意味を含む言葉です。ここでは「コンピュータの構成要素」として解説します。

コンピュータは、ハードウェアだけでは何も実行してはくれません。ハードウェアに対して「何々をしなさい」という指令をして、はじめてコンピュータは仕事をします。この指令のことをソフトウェアといいます。

ソフトウェアも幅広い意味をもつ言葉で、「指令データ」そのものを意味することもあれば「指令データを作成し、コンピュータを利用する技術」を意味することもあります。

ハードウェアとソフトウェアを統合してコンピュータを使うとき、これらをまとめて「システム技術」といいます。

コンピュータの概念を右に示します。



COMET

情報技術者試験のアセンブラ言語CASLの動く仮想コンピュータCOMETは、ハードウェア構成について特に指定はなく、各処理系に委ねられています。本機のCASLモードは、試験の案内書に掲載されている仕様書を実現したものです。

CASL

仮想コンピュータCOMETで動くアセンブラ言語がCASLです。それでは、これからCASLについて説明していきましょう。

COMETの構成

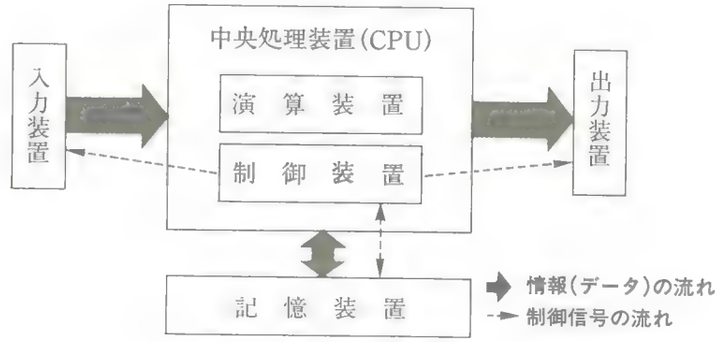
ここで学ぶこと
☆コンピュータの構成
☆中央処理装置の構成
☆記憶装置
☆入出力装置

まとめ
◎一般にコンピュータは中央処理装置、入出力装置、記憶装置から構成されています。
◎中央処理装置では実際の計算を行ないます。
◎記憶装置にはプログラムとデータが保存されます。
◎中央処理装置は入出力装置を通して外部と対話します。

コンピュータの構成

コンピュータは、一般に中央処理装置、出力装置、記憶装置から構成されています。そして、中央処理装置は、演算装置、レジスタ、制御装置から構成されています。次にコンピュータの構成を図示します。このようなハードウェア構成を持つものをCOMETと呼びます。

コンピュータの構成



中央処理装置の構成

中央処理装置はCPU（Central Processing Unit）と呼ばれ、コンピュータシステムの中核を構成します。中央処理装置は、制御装置、演算装置、レジスタから構成されています。レジスタについては後のページで説明します。

制御装置	記憶装置に記録されている命令を読み出し解読して、演算装置や主記憶装置、入出力装置に信号を送ります。このとき、各装置のタイミングを取って動作の制御を行ないます。
演算装置	制御装置によって解読されたプログラムの指令にしたがって、判断や演算を行ないます。

記憶装置

記憶装置には、主記憶装置と補助記憶装置の２種類があります。

主記憶装置は、実行しているプログラムやデータを記憶する場所で、それらプログラムやデータを中央処理装置とやりとりします。本機では、メインメモリーにあたります。

補助記憶装置は、限界のある主記憶装置の容量を補助する役割を持っています。

入出力装置

プログラムやデータ、演算結果などを入出力するための装置です。

本機では、本体のキーボードが入力装置に本体の液晶表示や外部に接続できるプリンタなどが出力装置にあたります。

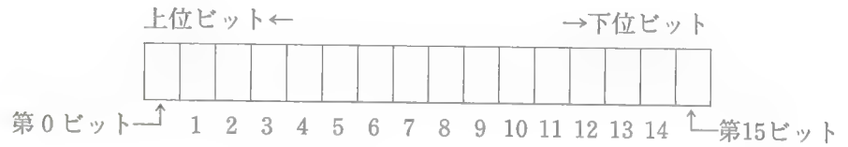
データの構成とレジスタ

ここで学ぶこと
☆データの構成
☆COMETのレジスタとカウンタ
☆汎用レジスタ
☆フラグレジスタ
☆プログラムカウンタ

まとめ
◎COMETではデータは、1語＝16ビット単位で扱われます。
◎COMETには、3種類のレジスタが用意されています。
◎汎用レジスタは様々な計算用に使われます。
◎フラグレジスタは、演算結果の符号を保持します。
◎プログラムカウンタは次に実行すべき命令の先頭アドレスを示します。

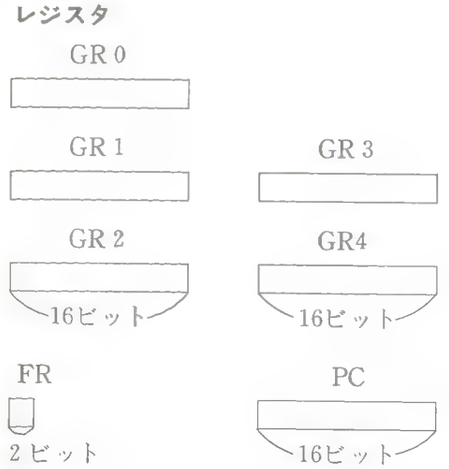
データの構成

COMETは１語16ビット単位のコンピュータで、１語は次のような構成になっています。



COMETのレジスタとカウンタ

COMETには、汎用レジスタ、フラグレジスタ、プログラムカウンタの３種類のレジスタが用意されています。



汎用レジスタ(General Resister 略称GR)

汎用レジスタは、16ビットの大きさを持つレジスタでGR0, GR1, GR2, GR3, GR4の5個があります。

汎用レジスタには以下のような機能があり、それぞれ使用できるレジスタが決められています。

演算用レジスタ(GR)	算術演算、論理演算、比較演算、シフト演算などに用います。GR0, GR1, GR2, GR3, GR4のすべてが使えます。
インデックスレジスタ(XR)	アドレス修飾(後述)に使います。GR1, GR2, GR3, GR4の4つが使えます。
スタックポインタ(SP)	スタック(後述)のアドレスを指定します。スタックポインタとしてはGR4だけが使えます。

フラグレジスタ(Flag Register 略称FR)

FRはCOMET内に1個だけあり、2ビットで構成されており、演算結果の符号を保持しています。(右表)また比較演算命令(CPA, CPL)のとき、FRはGRの内容と比較対象を比べた結果によって決まります。

演算結果 比較結果	正 >	ゼロ =	負 <
FRの値(2進)	00	01	10
本機の表示	0	1	2

プログラムカウンタ(Program Counter 略称PC)

PCは、次に実行すべき命令の先頭アドレスを格納する16ビットの大きさのレジスタで、COMET内に1個あります。

命令語の構成

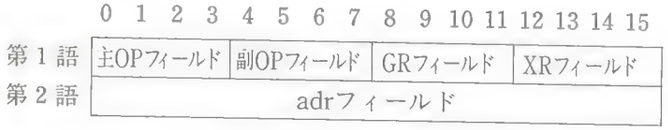
ここで学ぶこと
☆命令語の構成

まとめ
◎命令語は一つが32ビットの大きさを持っています。

命令語の構成

命令語は32ビットの大きさ、すなわち2語の長さを持っています。その構成は以下のようになっています。

命令語の構成



OPフィールド	8ビットで構成されていて、命令コードの種類を表わします。
GRフィールド	4ビットで構成されていて、命令の対象となる汎用レジスタを指定します。
XRフィールド	4ビットで構成されていて、アドレス修飾するためのインデックスレジスタの番号を指定します。
adrフィールド	16ビットで構成されていて、命令の対象となる主記憶装置上のアドレスを指定します。また、定数を表わすこともあります。シフト命令の時には、シフトするビット数を示します。

下記の表は、各命令語を、アセンブリ言語から機械語に変換した例の一部です。試験センターの仕様書では変換方法を規定していませんので、本機では独自に定めています。

アセンブリ言語	第1語(16進) OP	GR	XR	第2語(16進) adr
LD GR0, 10, GR1	1	0	1	000A
SUB GR1, −1, GR2	2	1	2	FFFF
AND GR0, 10	3	0	0	000A
CPL GR1, −1, GR2	4	1	2	FFFF
SLA GR0, 10, GR1	5	0	1	000A
JPZ 10, GR1	6	0	1	000A
PUSH −1, GR3	7	0	3	FFFF
POP GR3	7	1	3	0000
CALL 10, GR1	8	0	1	000A
RET	8	1	0	0000

アドレス修飾と実効アドレス

ここで学ぶこと
☆アドレス
☆実効アドレス

まとめ
◎主記憶装置の1語1語につけられた通し番号をアドレス(番地)と呼びます。
◎命令語のXRの内容とadrをアドレス加算したものが、実効アドレスです。

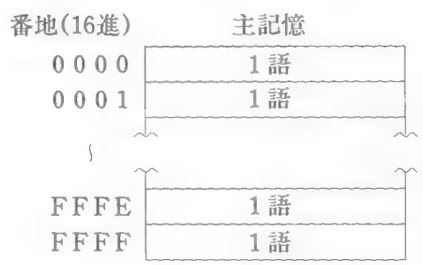
アドレス

主記憶装置は、16ビット1語を最大2^{16}＝65536語まで装備できます。

アドレスは、0から65535までの整数値で与えます。アドレスの表記は通常16進数で0000からFFFFまでとなります。

COMETのアドレス空間を右図に示します。

COMETのアドレス空間



中央処理装置は、主記憶装置内の命令を解読し①データのやりとりを行なう、または②実行制御を移す（ジャンプする）ためにアドレスを指定しなければなりません。

実効アドレス

命令を解読して実行するとき、その対象となるアドレスは、命令語のXRフィールドとadrフィールドで指定します。これをアドレス修飾といい、指定されたアドレスを「実効アドレス」といいます。

実効アドレスは、XRレジスタの値とadrの値のアドレス加算で計算されます。

例　　LD　GR0, 10, GR1

この例ではXRがGR1、adrが10になっています。したがって、この場合は「GR1の内容+10」が実効アドレスとなります。

もしGR1の内容が3ならば、実効アドレスは3＋10で13になります。

ところが、GR1の内容がFFFA（16進）の場合はどうでしょうか。実際に加算してみると、

FFFA＋000A（10進数で10）＝10004

となりますが、5桁目の1が無視されて0004が実効アドレスになります。このように5桁目にあふれた数を無視する加算を「アドレス加算」といいます。

XRフィールドが省略されているときはadrの値がそのまま実効アドレスになります。

スタック

ここで学ぶこと
☆スタック
☆スタックポインタ

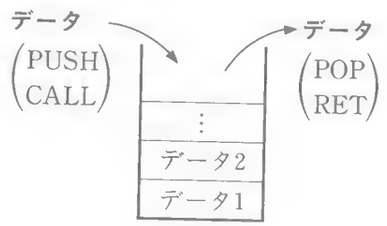
まとめ
◎スタックは、レジスタやメモリー内容の一時退避や、呼び出したサブルーチンから復帰するための戻り番地の保存に使います。
◎スタックに保存されているデータのトップ番地を示しているのがスタックポインタです。

スタック

スタックは、レジスタやメモリー内容の一時退避や、呼び出したサブルーチンから復帰するための戻り番地の保存に使います。

スタックはいわばデータの井戸のようなもの（下図参照）で、スタックにデータを入れるとそれがどんどんつみ上がっていく形になります。したがって、最後に入れたデータを最初に出すことができ、最初に入れたデータが最後に出されることになります。

スタックにデータを入れる命令として、PUSHとCALL、それに対応してスタックからデータを出す命令として、POPとRETがあります。



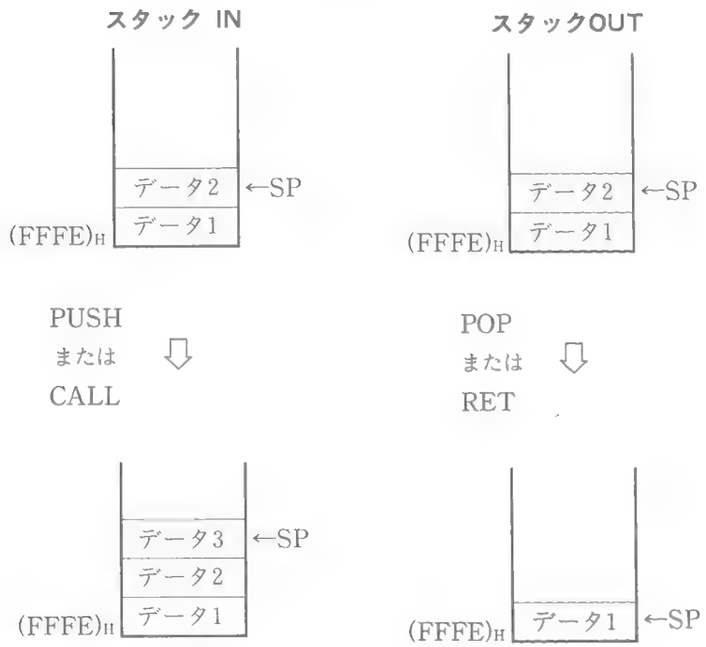
スタックポインタ(SP)

汎用レジスタGR4はスタックポインタとして使われます。

スタックポインタは、スタックのデータトップの位置を示しています。その初期値はFFFF(16進)になっています。

スタックポインタの内容は、PUSH命令、POP命令、CALL命令、RET命令によって変化します。

スタックポインタの模式図



キャラクターセット

ここで学ぶこと
☆COMETのキャラクターセット

まとめ

◎COMETでは英数字、カナ文字、図形文字が使用でき、それぞれコード番号を持っています。

キャラクターセット

COMETはJIS X0102ローマ字・カタカナ用8単位符合で規定するキャラクターセットをもちます。

本機では、下表のキャラクターが使用できます。

情報処理試験においてはJIS X0102の表の一部だけが示されています。試験問題において、表にない文字が必要なときは、その文字のビット構成が問題文の中で与えられます。

8単位符号の構成

b8	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
b7	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
b6	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
b5	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
b4 b3 b2 b1 行 / 列	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
0 0 0 0	0	32個の機能キャラクターの領域	8P	94個の図形キャラクターの領域					32個の機能キャラクターの領域		10/0	94個の図形キャラクターの領域				
0 0 0 1	1															
0 0 1 0	2															
0 0 1 1	3															
0 1 0 0	4															
0 1 0 1	5															
0 1 1 0	6															
0 1 1 1	7															
1 0 0 0	8															
1 0 0 1	9															
1 0 1 0	10															
1 0 1 1	11															
1 1 0 0	12															
1 1 0 1	13															
1 1 1 0	14															
1 1 1 1	15															15/15

備考：図中のb1～b8は、8単位符号ビットの組み合わせを示す。

ローマ字・片仮名用8単位符号

列 / 行	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
0	NUL	TC7(BLE)	(SP)	0	@	P	'	p	⋮	⋮	未定義	ー	タ	ミ	⋮	⋮
1	TC1(SOH)	DC1	!	1	A	Q	a	q	⋮	⋮	。	ア	チ	ム	⋮	⋮
2	TC2(STX)	DC2	〃	2	B	R	b	r	⋮	⋮	「	イ	ツ	メ	⋮	⋮
3	TC3(ETX)	DC3	#	3	C	S	c	s	⋮	⋮	」	ウ	テ	モ	⋮	⋮
4	TC4(EOT)	DC4	$	4	D	T	d	t	⋮	⋮	、	エ	ト	ヤ	⋮	⋮
5	TC5(ENQ)	TC8(NAK)	%	5	E	U	e	u	⋮	⋮	・	オ	ナ	ユ	⋮	⋮
6	TC6(ACK)	TC9(SYN)	&	6	F	V	f	v	未	未	ヲ	カ	ニ	ヨ	未	未
7	BEL	TC10(ETB)	'	7	G	W	g	w	定	定	ア	キ	ヌ	ラ	定	定
8	FE0(BS)	CAN	(	8	H	X	h	x	義	義	イ	ク	ネ	リ	義	義
9	FE1(HT)	EM	)	9	I	Y	i	y	⋮	⋮	ウ	ケ	ノ	ル	⋮	⋮
10	FE2(LF)	SUB	*	:	J	Z	j	z	⋮	⋮	エ	コ	ハ	レ	⋮	⋮
11	FE3(VT)	ESC	+	;	K	[	k	{	⋮	⋮	オ	サ	ヒ	ロ	⋮	⋮
12	FE4(FF)	IS4(FS)	,	<	L	¥	ℓ	¦	⋮	⋮	ヤ	シ	フ	ワ	⋮	⋮
13	FE5(CR)	IS3(GS)	−	=	M	]	m	}	⋮	⋮	ユ	ス	ヘ	ン	⋮	⋮
14	SO	IS2(RS)	.	>	N	^	n	~	⋮	⋮	ヨ	セ	ホ	゛	⋮	⋮
15	SI	IS1(US)	/	?	O	_	o		⋮	⋮	ッ	ソ	マ	゜	⋮	未定義

オペレーティングシステム

ここで学ぶこと
☆オペレーティングシステム

まとめ
◎命令の実行に関するシステムを「オペレーティングシステム」といいます。

オペレーティングシステム

情報処理技術者試験のアセンブラ言語仕様書には、オペレーティングシステムについて次のように記載されています。本機のCASLもそれに準拠しています。

オペレーティングシステム

プログラムの実行に関して、利用者プログラムとオペレーティングシステムとの間に、次の取り決めがあります。

(1)プログラムの起動

プログラムはオペレーティングシステムにより起動されます。プログラムが格納される番地は不定とするが、(2)で述べるスタック領域も含めてプログラムの実行に支障を与えないものとします。

(2)スタック領域

オペレーティングシステムは、プログラムの起動時に、スタック領域を確保し、スタック領域の最後のアドレスに1をアドレス加算した値をSPに設定します。この領域は、プログラムでスタックとして利用されます。

スタック領域は、試験問題のプログラムで使用するのに十分な容量が確保されているものとします。

(3)入出力装置の割当て

IN命令に対応する入力装置、OUT命令に対応する出力装置の割当ては、プログラムの実行に先立ってオペレータが行ないます。入出力装置には、コンソール表示装置、タイプライター、カード読取り装置、カードせん孔装置などがあります。

(4)入出力処理

各種の入力装置からのデータの入出力は、IN、OUTマクロ命令で行ないますが、媒体や装置による入出力手続きの違いはすべてオペレーティングシステムが吸収し、システムの標準手続き（異常処理を含む）、標準形式で入出力を行ないます。したがって、このマクロ命令の使用者は、入出力装置の違いを意識する必要はありません。

(5)未定義ラベル

アセンブラは、機械語命令のオペランド中のラベル、DC命令のアドレス定数のラベルのうち、そのプログラム内で定義されていないラベルを、他のプログラムのSTART命令のラベル（他のプログラムの入口名）と解釈します。

この場合、アセンブラはアドレスの決定を保留し、その決定をオペレーティングシステムに任せます。オペレーティングシステムは、実行に先立って他のプログラムのSTART命令のラベルとの結合処理を行ないアドレスを決定します（プログラムの連結）。

命令の形式

ここで学ぶこと
☆ラベル
☆命令コード
☆オペランド
☆注釈(コメント)

まとめ
◎命令語は、ラベル欄、命令コード欄、オペランド欄、注釈欄から構成されています。
◎ラベルは、6文字以内の英数字です。
◎命令コードは、命令の種類を表わします。
◎オペランドは、命令コードの修飾語です。
◎コメントは、プログラムをわかりやすくします。

以下にソースプログラムのコーディング例を示します。

ラベル欄	命令コード欄	オペランド欄	注釈(コメント)欄
REI	START	STAT	;プログラム　ノ　ハジメ
STAT	OUT	CHR, N5	;モジレツ　ノ　シュツリョク
	EXIT		;ジッコウシュウリョウ
N5	DC	5	;テイスウ5
CHR	DC	'コンニチハ'	;モジレツ
	END		;プログラム　ノ　オワリ

ラベル

ラベル欄にはラベルを書きます。必要がなければ省略できます。
ラベルは、6文字以内の英数字で書かれなければなりません。
行の第1文字目から書かれている文字列は、ラベルとみなされます。
ラベルの先頭の文字は、英文字の大文字でなければなりません。第2文字以降は、英大文字、数字のいずれでもかまいません。
計算機の内部では、命令につけられたラベルは、その命令語または領域の先頭のアドレス番地を表わしています。

命令コード

命令コード欄には、命令コードを書きます。
ラベルをつけたときには、ラベルのあとに少なくとも1文字分以上の空白が必要です。
ラベルをつけないときは、第2文字以降の任意の位置から書き始めます。

オペランド

オペランド欄には、オペランドを書きます。
命令コードのあとに少なくとも1文字分の空白が必要です。最大72文字目までがオペランド欄として使用できます。次の行に続けて書くことはできません。

注釈

注釈欄には処理系の許す任意の文字でコメントを書くことができます。本機では、英文字とカナが使えます。
セミコロン「;」以降が注釈行となります。第1文字目に「;」があるときと、「;」以降に空白しかないときは、その行全体が注釈行となります。

命令の種類と表記法

ここで学ぶこと
☆命令の種類
☆命令の表記法

まとめ
◎擬似命令は、アセンブラに指示を与える命令です。
◎CASLの機械語命令は23種類あり、翻訳されると機械語は逐一対応されます。
◎マクロ命令は、入出力と終了のための命令です。
◎命令の表記にはさまざまな決まりがあります。

命令の種類

CASLの命令には大きく分類して、擬似命令、機械語命令、マクロ命令の3種類があります。

擬似命令

擬似命令は、アセンブラの制御、定数の定義、プログラムの連結に必要なデータ生成などを行ないます。擬似命令にはSTART、END、DC、DSの4種類があります。

機械語命令

機械語命令は、ハードウェアCOMETの命令で中央処理装置によってそのまま認識されて実行されます。機械語命令はLD、ST、LEAなど23種類あります。

マクロ命令

マクロ命令には、入出力(IN, OUT)と終了(EXIT)のための3種類の命令があります。

命令の表記法

次ページ以降の各命令の解説では、次の表記法を用います。

LABEL	ラベル欄に書くラベルです。
GR	汎用レジスタ(0≦GR≦4)、またはGRn(0≦n≦4)。
XR	インデックスレジスタ(1≦XR≦4)、またはGRx(1≦x≦4)。
SP	スタックポインタGR4。
adr	ラベル名または10進定数。 ラベル名のときはラベルに対応する番地（数値）を示します。 10進定数は次の範囲で使用できます。 -32768≦adr≦65535
実行アドレス	「adr」と「XRの内容」とのアドレス加算値。またはその値が示す番地。
(X)	X番地の内容。Xがレジスタの時はレジスタの内容です。
〔 〕	〔 〕に囲まれた部分が省略可能であることを示します。 XRを省略した場合、インデックスレジスタによる修飾は行なわれません。

命令一覧

START、END、DC、DS 擬似命令

命令名	開始　START	書　式
		LABEL START〔adr〕
機能	プログラムの先頭に必ず書かなければなりません。adrのラベル名の番地から実行され、省略されるとプログラムの先頭から実行されます。	

命令名	終了　END	書　式
		END
機能	プログラムの終了を表わします。プログラムの最後やサブルーチンの最後には必ず書かなければなりません。ラベル、オペランドはありません。	

メインプログラムの記述例

```
MAIN START
        プログラム
        EXIT
        END
```

サブルーチンの記述例

```
SUBR START
        プログラム
        RET
        END
```

メインプログラムを記述するときは、先頭にSTART命令を書き、次にプログラムの内容を書き最後に処理を終了させるEXITとENDを書きます。

サブルーチンも同様に、先頭にSTARTを最後にENDを書きます。

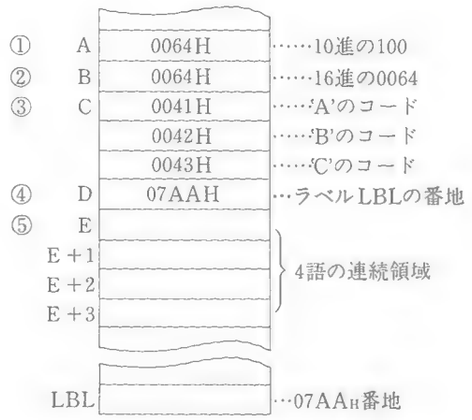
命令名	定数定義　Define Constant	書　式
		〔LABEL〕DC 定数
機能	メモリーに定数データを格納します。定数には、10進(-65535≦n≦65535)、16進(4桁)、文字定数、アドレス定数の四種類が使用できます。	

命令名	領域定義　Define Storage	書　式
		〔LABEL〕DS n
機能	n(≧0)によって指定した語数だけ連続領域を確保します。nが0のときは領域は確保されませんが、ラベルは有効です。	

DC、DS命令でのメモリー構成

DC、DS命令には下のように5種類の使い方があります。それぞれ以下のようなメモリー構成になります。

	ラベル	命令	オペランド
①	A	DC	100
②	B	DC	＃0064
③	C	DC	'ABC'
④	D	DC	LBL
⑤	E	DS	4

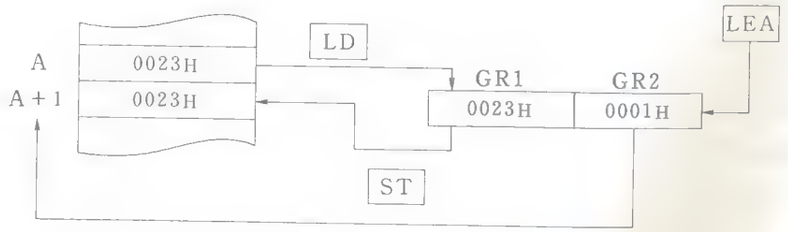


LD、ST、LEA 機械語命令:ロードストア命令、ロードアドレス命令

命令名	ロード Load	書　式
		〔LABEL〕 LD GR, adr 〔, XR〕
機能	実効アドレスによって示された番地の内容を、レジスタ(GR) に設定します。	
命令名	ストア STore	書　式
		〔LABEL〕 ST GR, adr 〔, XR〕
機能	レジスタ(GR)の内容を、実効アドレスによって示された番地に格納します。	
命令名	ロードアドレス Load Effective Address	書　式
		〔LABEL〕 LEA GR, adr 〔, XR〕
機能	実効アドレスによって示される番地の内容または10進定数を、レジスタに設定します。 また、このときの値によってFRの値を設定します。	

プログラム例

ラベル	命　令	オペランド	解　説
BGN	START		まず、A番地に格納されている数値がGR1に格納されます(LD)。次に、GR2に定数1が格納されます(LEA)。最後に、A+GR2の内容の番地、すなわちA＋1番地にGR1の内容35が格納されます(ST)。
	LD	GR1, A	
	LEA	GR2, 1	
	ST	GR1, A, GR2	
	EXIT		
A	DC	35	
	DS	1	
	END		



コラム●試験のCASL仕様書と本機のCASLの違い

VX-4のCASLは、試験センターのCASL仕様書に準拠して作成されていますが、実際にプログラムをシミュレートする都合のため下記の点で追加、変更しています。

項　目	試験センターのCASL仕様書	VX-4のCASL
アドレス空間	COMETは、1語16ビットの計算機であって、アクセスできるアドレスは0番地から65535番地までです。	メモリーに制限があるため、オブジェクトで確保された範囲内となります。この範囲を超えたアクセスはOut of addressエラーとなります。
プログラムの起動	プログラムはオペレーティングシステムにより起動されます。プログラムが格納される番地は不定とします。	オブジェクトプログラムは、アドレス0番地から格納されます。
スタック領域	プログラムの起動時に、オペレーティングシステムは、スタック領域を確保し、スタック領域の最後のアドレスに1をアドレス加算した値をSPに設定します。この領域は、プログラムでスタックとして利用されます。スタック領域は、試験問題のプログラムで使用するのに十分な容量が確保されているものとします。	スタック領域は、アドレスFFFFH番地からFF80H番地まで確保しています。ただし、使用可能領域は、FFFEH番地からFF80H番地までです。
入出力装置の割り当て	IN命令に対応する入力装置、OUT命令に対応する出力装置の割り当てはプログラムの実行に先立ってオペレータが行ないます。入出力装置には、コンソール表示装置、タイプライター、カード読み取り装置、カードせん孔装置などがあります。	IN命令に対応する入力装置はキーボード、OUT命令に対応する出力装置は表示画面およびプリンタと指定されています。
DC定数	定数で指定した10進数値を1語の2進数データとして格納します。ただし、定数が−32768～32767の範囲にないときは、その下位16ビットを格納します。	定数として10進数値を用いる場合、−65535～65535が格納可能です。

ADD、SUB 機械語命令：算術演算命令

命令名	算術加算 ADD arithmetic	書　式
		〔LABEL〕ADD GR, adr〔, XR〕
機能	GRの内容と実効アドレスによって示される番地の内容を算術加算してその結果をGRに設定します。演算結果によって、FRを設定します。	

命令名	算術減算 SUBtract arithmetic	書　式
		〔LABEL〕SUB GR, adr〔, XR〕
機能	GRの内容と実効アドレスによって示される番地の内容を算術減算してその結果をGRに設定します。演算結果によって、FRを設定します。	

プログラム例

ラベル	命　令	オペランド	解　説
BGN	START		まず、A番地に格納されている16進数値＃0029がGR1に設定されます。次に、この値にB番地の内容である＃000Eを算術加算した値＃0037が、GR1に格納されます(ADD)。同様に、C番地の内容である#001AをGR1から算術減算した値＃001Dが、GR1に格納されます。(SUB) 最後に、GR1の内容をANS番地に格納します。 0029 ＋) 000E　(ADD) 0037 −) 001A　(SUB) 001D
	LD	GR1, A	
	ADD	GR1, B	
	SUB	GR1, C	
	ST	GR1, ANS	
	EXIT		
A	DC	＃0029	
B	DC	＃000E	
C	DC	＃001A	
ANS	DS	1	
	END		

AND、OR、EOR 機械語命令： 論理演算命令

命令名	論理積 AND	書　　式
		〔LABEL〕 AND GR, adr 〔, XR〕
機能	GRの内容と実効アドレスによって示される番地の内容のビットごとの論理積を、GRに設定します。演算結果によって、FRを設定します。	
命令名	論理和 OR	書　　式
		〔LABEL〕 OR GR, adr 〔, XR〕
機能	GRの内容と実効アドレスによって示される番地の内容のビットごとの論理和を、GRに設定します。演算結果によって、FRを設定します。	
命令名	排他的論理和 Exclusive OR	書　　式
		〔LABEL〕 EOR GR, adr 〔, XR〕
機能	GRの内容と実効アドレスによって示される番地の内容のビットごとの排他的論理和を、GRに設定します。演算結果によって、FRを設定します。	

プログラム例

ラベル	命　令	オペランド	解　説
BGN	START		まず、GR1に16進定数＃5555が格納されます。次に、それとA番地の内容である＃137Fと論理積を取った結果＃1155がGR1に格納されます。 ＃5555 ＝ 0101 0101 0101 0101 AND) ＃137F ＝ 0001 0011 0111 1111 ＃1155 ＝ 0001 0001 0101 0101 (このプログラム例でのANDと同様にORとEORも使用できます。)
	LD	GR1, A	
	AND	GR1, B	
	EXIT		
A	DC	＃5555	
B	DC	＃137F	
	END		

コラム●論理演算●

論理積(AND)では2つの値とも1のときのみ1になり、論理和(OR)ではどちらか一方が1ならば1になり、排他的論理和(EOR)では2つの値が異なるときのみ1になります。これを図で表わすと右のようになります。

演算値		AND	OR	EOR
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

CPA、CPL 機械語命令：比較演算命令

命令名	算術比較 ComPare Arithmetic	書　　式
		〔LABEL〕 CPA GR, adr 〔, XR〕
機能	GRの内容と実効アドレスによって示される番地の内容を算術比較し、その結果により、FRの値を設定します。	
命令名	論理比較 ComPare Logical	書　　式
		〔LABEL〕 CPL GR, adr 〔, XR〕
機能	GRの内容と実効アドレスによって示される番地の内容を論理比較し、その結果により、FRの値を設定します。	

プログラム例

ラベル	命　令	オペランド	解　説
BGN	START		まず、GR1に定数－32が格納されます。次に、CPA命令によりA番地の内容＃20＝$(32)_{10}$と算術比較されます。この場合は、GR1$(-32)_{10}$＜A番地の内容$(32)_{10}$なのでFRのビット値は$(10)_2$になります。 同様に、次のCPL命令でも比較が行われますが、ここでは以下に示したようなビット構成による論理比較が行なわれます。この場合は、 GR1$(FFE0)_{16}$＞A番地の内容$(0020)_{16}$なのでFRのビット値は$(00)_2$になります。 GR1＝$(-32)_{10}$の ビット構成　1111 1111 1110 0000 A番地の内容＃20＝$(32)_{10}$の ビット構成　0000 0000 0010 0000
	LEA	GR1, －32	
	CPA	GR1, A	
	CPL	GR1, A	
	EXIT		
A	DC	＃0020	
	END		

SLA、SRA、SLL、SRL　機械語命令:シフト演算命令

命令名	算術左シフト Shift Left Arithmetic	書　　式
		〔LABEL〕 SLA GR, adr 〔, XR〕
機能	GRの内容を実効アドレスの数だけ左にシフトします。ただし、最上位ビット（符号ビット）はシフト前の値が保存され、空きビットには0が入ります。	
命令名	算術右シフト Shift Right Arithmetic	書　　式
		〔LABEL〕 SRA GR, adr 〔, XR〕
機能	GRの内容を実効アドレスの数だけ右にシフトします。ただし、最上位ビットはシフト前の値が保存され、その値で空きビットが埋められます。	
命令名	論理左シフト Shift Left Logical	書　　式
		〔LABEL〕 SLL GR, adr 〔XR〕〔, XR〕
機能	GRの内容を実効アドレスで示された数だけ左にシフトします。 シフトの結果による空きビットには0が入ります。	
命令名	論理右シフト Shift Right Logical	書　　式
		〔LABEL〕 SRL GR, adr 〔, XR〕
機能	GRの内容を実効アドレスで示された数だけ右にシフトします。 シフトの結果による空きビットには0が入ります。	

プログラム例

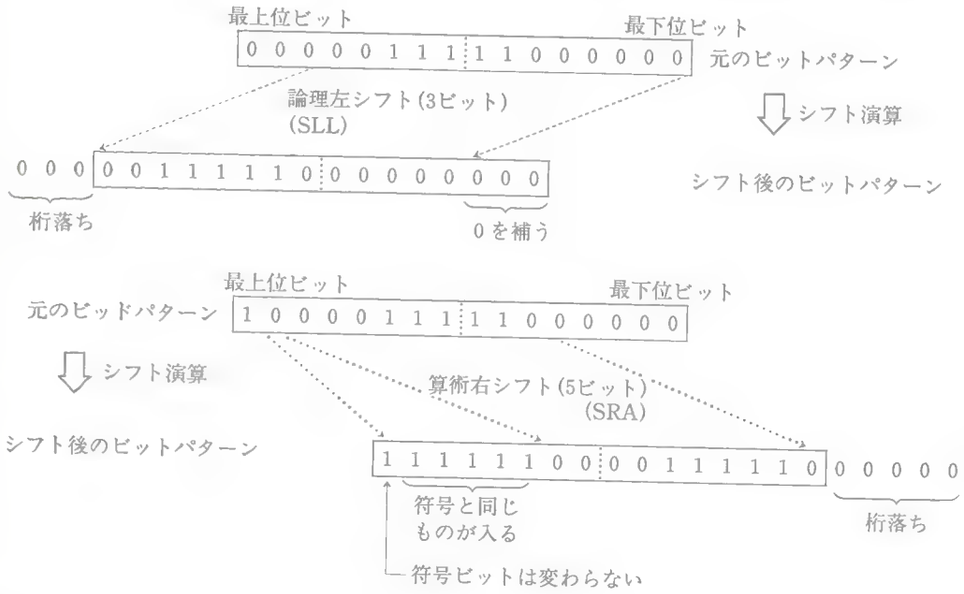
ラベル	命　令	オペランド	解　説
BGN	START		まず、GR1にA番地の内容である7FFF(16進)が格納されます。次にGR1の内容が左に5桁算術シフトされます。その結果GR1には7FE0(16進)が残ります。フラグは正ですので$(00)_2$になります。 7FFF ＝ 0111 1111 1111 1111 0111 1111 1110 0000 ＝7FE0
	LD	GR1, A	
	SLA	GR1, 5	
	EXIT		
A	DC	#7FFF	
	END		

まとめ

GR1＝1110 0001 0101 0111のとき、
SLA GR1, 5ならば、GR1＝1010 1010 1110 0000となります。(左算術シフト)
SRA GR1, 5ならば、GR1＝1111 1111 0000 1010となります。(右算術シフト)
SLL GR1, 5ならば、GR1＝0010 1010 1110 0000となります。(左論理シフト)
SRL GR1, 5ならば、GR1＝0000 0111 0000 1010となります。(右論理シフト)
算術シフトの場合は、符号ビットは変化しません。
論理シフトの場合は、符号ビットは無視されてシフトします。
シフト命令実行後の値についているアンダーラインは実行前のGR1の値の中で保存されているものを表わしています。

コラム●シフト演算●

コンピュータの内部では、データは、2進数で表現されています。その2進数を16桁なり32桁なりを集めて、データのひとまとまりとして扱っています。このひとまとまりを16ビットまたは32ビットといいます。ひとまとまりになったデータを右方向あるいは左方向に移動する演算が、シフト演算です。



JPZ、JMI、JNZ、JZE、JMP　機械語命令:分岐命令

命令名	正分岐 Jump on Plus or Zero	書　　式 〔LABEL〕JPZ adr〔,XR〕
機能	FRのビット値が$(00)_2$「正」か$(01)_2$「零」のとき、実効アドレスに分岐します。それ以外のときは、次の命令に進みます。	
命令名	負分岐 Jump on MInus	書　　式 〔LABEL〕JMI adr〔,XR〕
機能	FRのビット値が$(10)_2$「負」のとき、実効アドレスに分岐します。それ以外のときは、次の命令に進みます。	
命令名	非零分岐 Jump on Non Zero	書　　式 〔LABEL〕JNZ adr〔,XR〕
機能	FRのビット値が$(00)_2$「正」か$(10)_2$「負」のとき、実効アドレスに分岐します。それ以外のときは、次の命令に進みます。	
命令名	雰分岐 Jump on ZEro	書　　式 〔LABEL〕JZE adr〔,XR〕
機能	FRのビット値が$(01)_2$「零」のとき、実効アドレスに分岐します。それ以外のときは、次の命令に進みます。	
命令名	無条件分岐 unconditional JuMP	書　　式 〔LABEL〕JMP adr〔,XR〕
機能	FRのビット値に関わらず、無条件に実効アドレスに分岐します。	

コラム●フラグ●

分岐命令は、フラグの値を参照してそのジャンプ先を決定します。
そのフラグの値を定める要因となる命令は以下の通りです。

- LEA、ADD、SUB、AND、OR、EOR、SLA、SLL、SRL命令による演算結果の符号(正$(00)_2$、零$(01)_2$、負$(10)_2$)
- CPA、CPL命令による比較の結果(GRの内容の方が実行アドレスの内容よりも大きい$(00)_2$、同じ$(01)_2$、小さい$(10)_2$)

プログラム例

ラベル	命　令	オペランド	解　説
BGN	START		このプログラムは、複数のデータを読み込み、そのデータの中から正の最小値を探し、そのデータを保存するというものです。ただし、「3」というデータがある場合は、そのデータだけを除外して処理を行ないます。 はじめに、CASLであつかえる符号つきの最大値7FFFHをMIN番地に格納しておきます。DATA番地からのデータをGR0に読み込み、それが、 ・A番地の内容1より小さい(0や負のデータは切り捨てる)。 ・B番地の内容3に等しい(3は除外する)。 ・MIN番地の内容より大きいか等しい。 とき、L2にジャンプします。そうでないときはGR0の内容をMINに格納します。L2以下では、GR1を1だけ増加して次のデータを読み込めるようにします。GR1がC番地の内容(5＝データ数)より小さいときはL1にジャンプして再び比較を行ないます。 最後に3以外の正の最小値4がMIN番地に格納されてプログラムが終了します。
	LD	GR0, MAX	
	ST	GR0, MIN	
	LEA	GR1, 0	
L1	LD	GR0,DATA,GR1	
	CPA	GR0, A	
	JMI	L2	
	CPA	GR0, B	
	JZE	L2	
	CPA	GR0,MIN	
	JPZ	L2	
	ST	GR0,MIN	
L2	LEA	GR1,1,GR1	
	CPA	GR1,C	
	JMI	L1	
	EXIT		
A	DC	1	
B	DC	3	
C	DC	5	
MAX	DC	＃7FFF	
MIN	DS	1	
DATA	DC	5	
	DC	－1	
	DC	0	
	DC	3	
	DC	4	
	END		

PUSH、POP　機械語命令:スタック操作命令
CALL、RET　機械語命令:コールリターン命令

命令名	プッシュ PUSH effective address	書　式 〔LABEL〕PUSH adr〔,XR〕
機能	スタックポインタ(SP)から1をアドレス減算したあと、実効アドレスによって示された番地をSPが示す番地に格納します。	
命令名	ポップ POP up	書　式 〔LABEL〕POP GR
機能	スタックポインタ(SP)の示す番地の内容をGRに設定し、SPに1をアドレス加算します。	
命令名	コール CALL subroutine	書　式 〔LABEL〕CALL adr〔,XR〕
機能	実効アドレスに示される番地に分岐し、処理の流れがサブルーチンに移されます。このとき、戻り番地がスタックに保存されます。	
命令名	リターン RETurn from subroutine	書　式 〔LABEL〕RET
機能	スタックに保存されていた戻り番地がプログラムカウンタにセットされ、サブルーチンからメインプログラムに処理の流れが戻されます。	

プログラム例

ラベル	命　令	オペランド	解　説
MAIN	START		まず、メインプログラムはサブルーチンを呼び出すだけの構成になっています。サブルーチンではPUSH、POP命令を行ない、スタックを通してGR1に100を入れます。そして、RET命令によってメインプログラムに戻ります。
	CALL	SUBR	
	EXIT		
	END		
SUBR	START		
	PUSH	100	
	POP	GR1	
	RET		
	END		

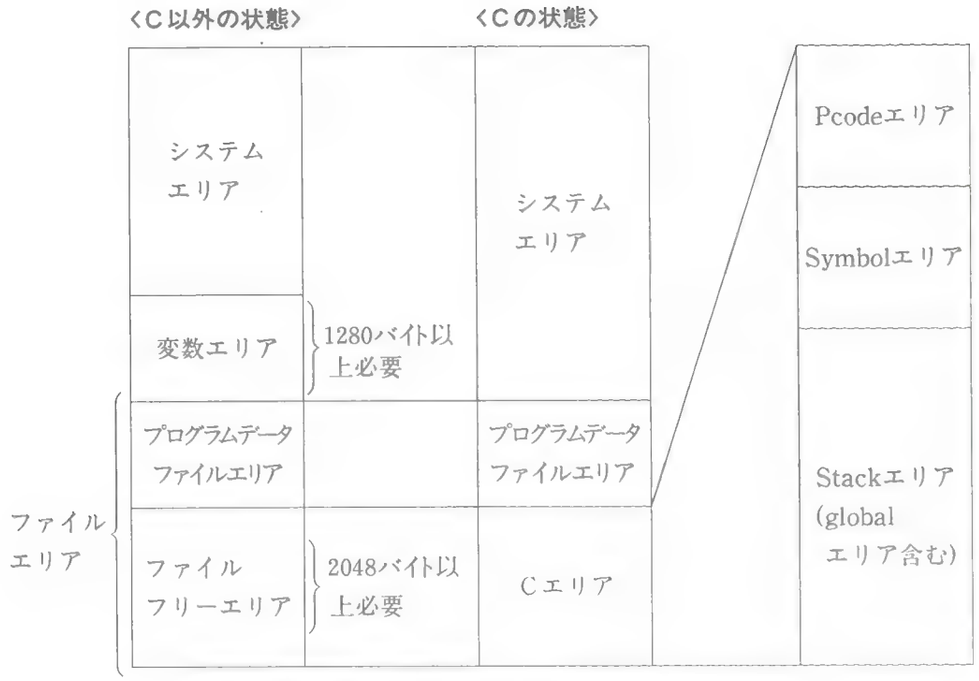
IN、OUT、EXIT　マクロ命令

命令名	IN 命令	書　式 〔LABEL〕IN 入力領域、入力文字長
機能	キーボードから1レコードの文字データを入力し、80語の入力領域に書いたラベル名の番地から格納します。入力文字長には、文字数が格納されます。	
命令名	OUT 命令	書　式 〔LABEL〕OUT 出力領域、出力文字長
機能	出力領域に格納されている文字データを、表示画面に1レコードとして出力します。ただし、出力文字長の長さまでしか出力しません。	
命令名	EXIT 命令	書　式 〔LABEL〕EXIT
機能	メインプログラムの実行を終了します。	

プログラム例

ラベル	命　令	オペランド	解　説
BGN	START		まず、A番地に80語の領域をとり、IN命令によりキーボードから文字列がA番地から一文字ずつ順番に格納され、B番地には文字数が格納されます。次に、OUT命令によりA番地から格納されている文字列を、B番地に示す文字数だけ読み出し、画面に表示します。 最後にEXIT命令でプログラムが終了します。
	IN	A, B	
	OUT	A, B	
	EXIT		
A	DS	80	
B	DS	1	
	END		

メモリーマップ



● C使用時のユーザーズエリア(プログラム・データエリア＋Cエリア)

最大　8KB RAM時　3611バイト

40KB RAM時　36379バイト

※最大とは、CLEAR，1280バイトの時です。

デフォルトは 8～16KB時　CLEAR，1536

40KB時　CLEAR，8192　です。

● Cを動作するには、変数エリアが最低1280バイトファイルフリーエリアが最低2048バイト必要です。

C言語エラーメッセージ一覧表

エラーメッセージ	エラー内容	対処方法
Illegal command	コマンドの書式に誤りがあります。	コマンドの書式を再確認してください。
Buffer overflow	256文字以上入力し、 IN キーを押した。	1行255文字以下にしてください。
Syntax error	構文エラーです。 配列の要素数に定数以外を指定しました。 定義されていない識別子または内部システムで使用している関数と同じ名前を使用しました。	構文及び識別子を確認してください。
Unknown character	識別子などに本機では許されていない文字が使用されました。	使用している文字を確認してください。
Illegal char constant	文字定数が1文字の表示可能文字、またはエスケープシーケンス以外です。 文字定数がシングルクォーティションでくくられていません。	文字定数を確認してください。
Illegal string	文字定数がダブルクォーティションでくくられていません。 文字列が255文字を超えています。	文字列を確認してください。
Illegal escape sequence	¥の後の文字が有効なエスケープシーケンスでありません。	エスケープシーケンスを確認してください。使用できるエスケープシーケンスは¥'・¥"・¥¥・¥ddd・¥xdd・¥a(07h)・¥b(08h)¥f(0Ch)・¥n(0Ah)・¥r(0Dh)・¥t(09h)・¥0(00h)です。
Arithmetic overflow	浮動小数点定数が、表現できる値の範囲を超えました。	浮動小数点定数を確認してください。
Illegal preprocessor	プリプロセッサの構文エラーです。	#define文を確認してください。
Illegal storage class	この部分では、指定の記憶クラスは使用できません。	記憶クラスを確認するか、指定位置を確認してください。
Illegal type	指定できない型指定子を使用しました。 データ定義で型指定子にvoidを指定しました。	型指定子を確認してください。

エラーメッセージ	エラー内容	対処方法
Illegal pointer DCL	ポインタ宣言は、＊を１つだけしか指定できません。 型指定子voidのポインタ宣言はできません。	ポインタ宣言を確認してください。
Illegal function DCL	キーワード、ライブラリー関数及び内部システムで使用している関数と同じ名前で関数定義や宣言はできません。	関数定義または宣言を確認してください。 別の名前を使用してください。
Illegal array DCL	配列宣言で要素数の指定が整数値でありません。 配列のサイズが大きすぎます。(65535バイトを超えています) 最初の次元以外の次元の要素数の指定がありません。 サイズを取っていない配列をローカル変数として宣言しようとしました。	配列宣言を確認してください。
Illegal initialization	関数は初期化できません。	初期化を確認してください。
Illegal argument DCL	関数は仮引数として宣言できません。 関数の仮引数中にない引数が、宣言されています。	関数定義の引数宣言を確認してください。
'識別子' undefined	識別子が未定義です。	識別子を確認してください。 定義されていなければ、その定義を追加してください。
'識別子' redefined	識別子が二重定義されています。	識別子を確認してください。 すでに定義されているものと同じ場合は、別の名前にしてください。
'識別子' not a function	関数として宣言されていない識別子を、関数として使おうとしました。	識別子を確認してください。
'識別子' not a variable	変数として宣言されていない識別子を、変数として使おうとしました。	識別子を確認してください。
Illegal sizeof	sizeofのオペランドは、識別子でなければなりません。	sizeofのオペランドを確認してください。
Illegal cast	void型にキャストできません。	キャストの型名を確認してください。
Illegal break	break文がdo、for、またはwhileいずれかの文中以外で使用されています。	break文はdo、for、またはwhile文の文中で使用するようにしてください。

エラーメッセージ	エラー内容	対処方法
Illegal continue	continue文がdo、for、またはwhileのいずれかの文中以外で使用されています。	continue文はdo、for、またはwhile文の文中で使用するようにしてください。
Too complex expression	式が複雑すぎます。	式を分割してください。
Too many nest	if、do、for、またはwhile文のネストの深さが25レベルを超えました。 複文のレベルが深すぎて、システムスタックがオーバーフローしました。	ネストを浅くしてください。
Too many break /continue	do，for，またはwhile文中のbreak文とcontinue文の合計が25を超えました。	break文とcontinue文の合計が25以内にしてください。
P-code overflow	Cエリア内のP-CODEエリアがオーバーフローしました。(プログラムが大きすぎます)	clear文を用いるか、RAMを増設しCエリアを増やしてください。
Symbol overflow	Cエリア内のSYMBOLエリアがオーバーフローしました。(識別子、定数が多すぎます)	clear文を用いるか、RAMを増設しCエリアを増やしてください。
'main' not found	main関数が見つかりません。	main関数を含むファイルをロードしてください。
Illegal 'main'	main関数に引数は指定できません。	main関数の引数をなくしてください。
'演算子' incompatible types	式中またはreturnの戻り値に互換性のない型があります。	オペランドの型を確認してください。
'演算子' illegal operand	演算子のオペランドの型が正しくありません。	オペランドの型を確認してください。
'演算子' illegal value	この演算子のオペランドは、左辺値でなければなりません。	オペランドを確認してください。
Illegal index	配列の添字の値が負、または定義されている要素数を超えました。	添字の指定を確認してください。
Illegal function call	関数を呼出す時、引数の個数が関数定義の仮引数の個数と一致しません。 または、引数の型を仮引数の型に変換できません。	引数の個数と型を確認してください。
Argument list too big	引数リストが、128バイトを超えました。	引数の数を少なくしてください。

エラーメッセージ	エラー内容	対処方法
Illegal stream pointer	ストリームポインタが正しくありません。 読み込みのためのストリームに書き込もうとしました。 書き込みのためのストリームから読み込もうとしました。	ストリームポインタを確認してください。
Mathematical arithmetic error	0による割り算などの数学的エラーです。 ライブラリー関数の引数が演算範囲を超えています。 float/double型のデータのフォーマットエラーです。	数値の値を確認してください。
Arithmetic overflow	計算結果が許される範囲を超えました。	計算式を計算してください。
Illegal argument	ライブラリー関数の引数が、許容範囲を超えています。	引数を確認してください。
Global overflow	Cエリア内のグローバルエリアがオーバーフローしました。(外部定義の変数と関数内の anto 変数の合計サイズが大きすぎます)	clear文を用いるか、RAMを増設しCエリアを増やします。
Stack overflow	Cエリア内のスタックエリアがオーバーフローしました。(再帰呼び出しなどにより、引数リストと内部定義の変数がとられているスタックが足りなくなりました)	clear文を用いるか、RAMを増設しCエリアを増やします。
Abort	ライブラリー関数abortにより実行が中断しました。	
Memory overflow	fileエリアがオーバーフローしました。	fileエリアの容量を増やしてください。fileフリーエリアは最低2048バイト必要です。
C underfind error	ハードウェア上の不具合、またはCプログラムを実行したときのメモリーの異常により発生する可能性があるエラーです。	ALL RESETボタンを押してください。
OM error	変数エリアが1280バイト以上ありません。	CLEAR,1536などとして1280バイト以上変数エリアをとってください。

索引

C言語

数字・記号

%文字……5、7
＊演算子……29
8進定数……10
10進定数……10
16進定数……10

アルファベット

abort……65
abs……70
acos……71
acosh……72
angle……73
asin……71
asinh……72
atan……71
atanh……72
beep……74
breakpt……65
call by value……35
calloc……66
char……17
clearerr……64
clrscr……74
cos……70
cosh……71
C言語モード……2
double……17
do～while……23
exit……65
exp……73
fflush……64
fgetc……56
fgets……58
float……16
for……22
fprintf……59
fputc……57
fputs……58
free……67
fscanf……62
getc……56
getch……64
getchar……56
gets……57
gotoxy……74
if……18
int……10
inport……64
log……73
log10……73
long……10、16
main……6、51
malloc……66
outport……64
pow……72
printf……7、59
putc……57
putchar……56
puts……58
RUN……54
scanf……9、62
sin……70
sinh……71
sprntf……59
sqrt……72
sscanf……62
strcat……68
strchr……69
strcmp……68
strcpy……67
strlen……67
tan……70
tanh……71
TROFF……54
TRON……54
while……22

50音

あ行

引数……7
インクリメント……22、25
エスケープシーケンス……8、40
演算子……42

か行

外部変数……35、47
型宣言……15、33
関数……51
関数定義……31
キーワード……36
記憶クラス……47
コメント……5、36

さ行

算術演算子……15
実行……3
実数……11
自動変数……35
書式変換仕様……60、62
シングルクォーテーション……6、12
数値関数……70
整数定数……41
静的変数……35

た行

代入演算子……15
ダブルクォーテーション……12
注釈……5
データ型……38
デクリメント……25

な行

内部変数……35、47
入出力関数……56
入力……3
ヌル(NULL)……40

は行

配列……27、49
配列の初期化……49
倍精度浮動小数点定数……41
比較演算子……20
ファイルエリア……3
複文……6
浮動小数点定数……11、41
ブレークモード……65
プリプロセッサ……53
プロセス関数……65
文……6
変数……6
変数の型……6
変数名……14
ポインタ……49
ポインタ変数……29

ま行

無限ループ……23
メモリー関数……66
文字定数……12、40
文字列関数……67
文字列定数……12、42

ら行

ライブラリー関数……51、55
ロード……3
論理演算子……21

BASIC
アルファベット

APPEND……101
BEEP……87、109
CHAIN……114
CLEAR……97、109
CLOSE……103、112
CLS……112
CONT……108
DATA……95、109
DEFCHR$……112
DEFSEG……112
DELETE……107
DIM……97、109
EDIT……81、84、107
ELSE……110
END……88、109
ERASE……109
FILES……113
FORMAT……113
FOR～NEXT……89
FOR～TO～STEP……109
GOSUB……93、110
GOTO……92、109
IF……88
IF～GOTO……110
IF～THEN……110
IF～THEN～ELSE……87
INPUT……86、101、110
INPUT#……103、113
KILL……113
LET……86、88、110
LINE INPUT#……114
LIST……81、83、106
LIST#……108
LOAD……107
LOAD#……108
LOCATE……112
MERGE……107
MERGE#……108
NAME……113
NEW……106
NEXT……109
ON ERROR GOTO……113
ON～GOSUB……111
ON～GOTO……111
OPEN……101、113
OUTPUT……101
PASS……106
POKE……112
PRINT……88、111
PRINT#……103、113
READ……95、109
READ#……112
REM……111
RENUM……106
RESTORE……96、109
RESTORE#……112
RESUME……113
RETURN……94、110
RUN……106
SAVE……106
SAVE#……108
SET……111
STAT……114
STAT CLEAR……114
STOP……85、111
SYSTEM……108
TROFF……112
TRON……112
VARLIST……112
VERIFY……107
WRITE#……112

50音
あ行

インタープリター……78
エディタ……82
エラーメッセージ……85

か行

行……76
行番号……76

さ行

サブルーチン……93、94、110、111
周辺機器……100
実行文……77
数値変数……105
セグメントベースアドレス……112
添え字……98

た行

データバンク……103
デバッグ……85
トレース……85

は行

配列……105、109
配列数値変数……105
配列変数……97、105
配列名……105、112
配列文字変数……105
バグ……85
パスワード……106
比較演算子……104
ファイル……101
ファイルディスクリプタ……102
文……76、104
プログラムエリア……78
変数……87、105
変数名……87、105、112

ま行

マニュアルコマンド……106
無限ループ……92
無条件ジャンプ……91
メインルーチン……94
文字演算子……104
文字データ領域……105
文字変数……105

CASL
アルファベット

ADD……147
AND……148
CALL……135、136、154
CASL……128
COMET……128
CPA……149
CPL……149
CPU……129
DC……143、144
DS……143、144
END……143
EOR……148
EXIT……155
FR……132
GR……131
IN……155
JMI……152
JMP……152
JNZ……152
JPZ……152
JZE……152
LD……145
LEA……145
OR……148
OUT……155
PC……132
POP……135、136、154
PUSH……135、136、154
RET……135、136、154
SLA……150
SLL……150
SP……136

SRA……150
SRL……150
ST……145
START……143
SUB……147

50音
あ行
アセンブラ……116
アセンブル……118
アドレス……134
アドレス加算……134
アドレス修飾……134
アルゴリズム……122
インデックスレジスタ……132
エラー……121
エラーメッセージ……121
オブジェクトプログラム……118、119
オペランド……116、141
オペレーティングシステム……139
か行
キャラクターセット……137
記憶装置……130
機械語命令……142
疑似命令……142
コールリターン命令……154
さ行
算術演算命令……147
シフト演算……151
シフト演算命令……150
実効アドレス……134
スタック……135
スタック操作命令……154
スタックポインタ……132、136
ソースプログラム……116
た行
ダンプ……123
中央処理装置……129
注釈……141
トレース……125
な行
入出力装置……130
は行
汎用レジスタ……132
比較演算命令……149
フラグ……152
フラグレジスタ……132
ブレークポイント……124
分岐命令……152
プログラムカウンタ……132
ま行
マクロ命令……142、155
命令一覧……143
命令コード……140
命令語……133
ら行
ラベル……116、140
レジスタ……124
ロードアドレス命令……145
ロードストア命令……145
論理演算……148
論理演算命令……148